DOCUMENT RESUME

ED 359 684 EC 302 244

AUTHOR	Burns, Edward
TITLE	A Manual for Single Switch and Adaptive Software Programming. Computer Applications for Students with Physical, Sensory, Developmental, and Learning Disabilities.
PUB DATE	[90]
NOTE	218p.
AVAILABLE FROM	Edward Burns, School of Education and Human Development, State University of New York at Binghamton, Binghamton, NY 13902-6000 (Apple program disk only, $5).
PUB TYPE	Guides - Non-Classroom Use (055) -- Computer Programs (101)
EDRS PRICE	MF01/PC09 Plus Postage.
DESCRIPTORS	*Assistive Devices (for Disabled); *Computer Software; *Disabilities; *Electronic Equipment; *Input Output Devices; *Programing
IDENTIFIERS	Apple Microcomputers; *Switches

ABSTRACT

This manual is intended as a guide and source of ideas for using single switches in adaptive software programming for people with disabilities who cannot use a traditional keyboard. The manual and associated program disk are comprised of over 100 programs, routines and files illustrating various uses of single switch and adaptive input devices. Programs were developed for use on the Apple family of computers and written in Applesoft BASIC. Complete program listings of programs on the disk are included in the manual. After an introduction, individual chapters address the following topics: (1) single switch fundamentals; (2) single switch input; (3) low-resolution graphics; (4) sound and speech; (5) single switch scan techniques; (6) high-resolution graphics; (7) single switch math; and (8) single switch reading. (Contains 17 references.) (DB)

****
*   Reproductions supplied by EDRS are the best that can be made   *
*   from the original document.   *
****

ED359684

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☐ This document has been reproduced as received from the person or organization originating it
☐ Minor changes have been made to improve reproduction quality
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

A Manual for Single Switch and Adaptive Software Programming

Computer Applications for Students with Physical, Sensory, developmental, and learning Disabilities

by

Edward Burns

State University of New York at Binghamton

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Edward Burns
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

EC 302244

BEST COPY AVAILABLE

Adaptive Software Programming

by

Edward Burns

State University of New York at Binghamton

CONTENTS

Introduction 4

1. Single Switch Fundamentals 9
2. Single Switch Input 32
3. Low-resolution Graphics 66
4. Sound and Speech 82
5. Single Switch Scan Techniques 116
6. High-resolution Graphics 133
7. Single Switch Math 170
8. Single Switch Reading 190

References 217

Introduction

Purpose

This manual should be used as a single switch idea book. The manual and program disk are comprised of over 100 programs, routines and files designed

The terms Apple® and ProDOS® are registered trademarks of Apple Computer, Inc. The terms Echo™, Echo IIb™ and Textalker™ are trademarks of Street Electronics Corporation.

Introduction

Purpose

This manual should be used as a single switch idea book. The manual and program disk are comprised of over 100 programs, routines and files designed to illustrate the many ways in which single switch and adaptive input devices can be used to meet the learning needs of students.

The programs and techniques described in the manual can be used in the following ways: 1) to understand the wide range of single switch and adaptive software applications available; 2) to design single switch software and to individualize programs to best accommodate specific learning needs; 3) to teach and illustrate single switch programming; and 4) to provide a source of readily available and documented single switch BASIC programs and programming techniques.

Because the Apple computer has been used extensively in the development of single switch software, various aspects of the Apple are considered in order to better design and use single switch software. In this respect, the manual and disk can be used as a toolkit to construct and modify single switch programs. The program disk contains two complete font sets as well as several authoring programs for constructing single switch assessment and learning and activities.

The manual and programs can be used by teachers, professionals and all those interested in understanding, modifying and individualizing single switch and adaptive input software. For persons unable to use a traditional keyboard, single switch software provides a means for meeting a variety of specific learning needs. By engaging a single switch device, software can be activated which develops readiness, promotes independent living skills, and teaches academic content such as reading, math and social studies.

Applications

The applications described in this manual include many single switch techniques, ranging from simple switch responses to matrix scanning, and various hardware applications (e.g., using the keyboard to provide switch input, the Touchwindow). Many different methods for recording switch responses. The following is a partial list of some of the single switch concepts considered in the manual:

Single Switch Input Techniques
Single Switch BASIC Programming
Array and Matrix Scanning
Controlled and Automatic Scanning
Creating Single Switch Language Boards
Developing Readiness Skills
Using and Creating Character Fonts
Using Low- and High-resolution Graphics
Single Switch Utilities
Authoring Systems
Sound and Music
Synthesized Speech
Using Large Type Displays
Creating Shape Tables
Morse Code
Single Switch Database Concepts
Single Switch Math and Reading
Using and Modifying Public Domain Software

IBM PC Single Switch Conversion Manual

An important goal of this manual is to provide information for better understanding single switch software, and to be able to individualize software to meet specific learning needs. Because many single switch software applications have been written in BASIC, knowing even a little BASIC will help to best meet individual learning needs. Although the manual does present many BASIC concepts for better understanding and using single switch software, an attempt has been made to present all BASIC information in a way that is directly related to single switch and adaptive software programming.

This manual provides an extremely diverse sampling of single switch BASIC applications. In addition, many program modifications and techniques for individualizing programs are presented. However, the purpose of this manual is not to provide a step-by-step approach for learning BASIC programming, but rather to understand how BASIC is used to create a wide range of single switch applications. This manual will to develop single switch software, and to consider the many ways in which BASIC can be used to meet the varied learning needs of the single switch user.

All of the programs are written in Applesoft BASIC and can be used with the Apple family of computers (II+, IIe, IIc or IIGS). The topics and single switch applications covered in this manual are intended to encourage participation. Enter and run the programs; experiment; make modifications; make changes that are deemed appropriate or useful.

Complete program listings are presented in the manual. These listings can be used as a basis for developing single switch programs, and to better understand single switch BASIC software design and applications.

Although single switch software is ideal for individual's in need of an adaptive input system, many of the applications can be used with students having diverse learning needs such as the developmentally disabled and student's with learning and/or emotional disorders. Because single switch software is often used in conjunction with scan routines and a speech synthesizer, many applications are suitable for the visually impaired and students having language disabilities.

An attempt has been made to include a variety of different, albeit easy-to-use, single switch activities, but there is no doubt that many important programming ideas are not presented. As a result, and as was said before, consider this an idea book. Improve the programming routines, and use whatever information is available to meet each individual's single switch needs.

Program Disk Sample

If the program disk is available, the following program will provide a general idea of the types of programs described in the manual and contained on the disk. As was already said, the programs cover a wide range of single switch topics and include many single switch utilities, routines for modifying and enhancing single switch programs and software ideas involving readiness, nonverbal, assessment, math and reading tasks.

To sample the program disk, insert the disk in drive #1, turn the Apple to ON, and enter **RUN FACE** after the screen prompt] and press RETURN. When the FACE program is run, each switch response causes an element of the face to appear on screen.

```
]RUN FACE
```

Now run the SWITCH FONT program. A word or phrase is first displayed using a large font character set. When the switch is engaged, feedback is given by re-displaying the word in various high-resolution colors. After the screen is cleared, the switch must be disengaged for several seconds before the next item is presented. The manual provides several ideas for modifying the program to meet individual student learning needs.

RUN SWITCH FONT

After the last of the five words in the SWITCH FONT program has been displayed, run the DISCRIM program. This is a three item scanning shape discrimination task. The object of the program is to select the shape that is different by engaging the switch when the scan cursor is beneath the shape that is different. As with most of the programs discussed in the manual, several modifications are given for individualizing programs.

RUN DISCRIM

The above three programs will provide some idea as to the types of single switch software contained in the manual and on the program disk. In addition to describing many single switch concepts, techniques are provided for adding sound, speech, and for creating and individualizing graphics, for adding speech and sound, and for designing software to teach reading, math and curriculum-based skills.

The overall goal is to provide a collection of programs and routines that will not only provided examples of many single switch software techniques, but to provided software ideas that can be readily changed to meet individual single switch learning needs.

Chapters and Disk Programs

The eight chapters in this manual describe over 100 single switch and adaptive software programs, as well as many routines to enhance and individualize programs. Although the chapters can be read in sequence, the manual can also be used as a reference for better understanding specific single switch concepts. The information and programs contained in each chapter are reasonable independent so that the ability to use one chapter does not necessarily require having read and studied the preceding chapters.

The first two chapters provide background information concerning the disk operating system, BASIC programming fundamentals, and single switch techniques. If you are familiar with programming, or simply are interested in learning more about single switch applications, skip chapters 1 and 2. However, as is frequently the case, just a little knowledge concerning BASIC programming will allow you an opportunity to really individualize many software applications.

The following groups these programs by chapter and specific single switch content area covered.

Chapter 1

Hello or greeting program:
HELLO

Chapter 2

Demonstration programs:
DEMO **ROCKET** **SWITCH COLOR** **TEST**

Utility Programs:
INPUT CHECK   HELLO-U   DISK MENU

Chapter 3

Low-resolution graphics:
LOWRES SAMPLE   LOWRES-SG   NILT   SCAN TRACKING
Low-resolution graphics applications:
ARCADE   JOYSTICK   JOYSTICK FEEDBACK   TOUCHWINDOW
LOWRES AUTHOR   LOWRES SCREEN   MEMORY MOVE
LOWRES SCREEN-2

Chapter 4

Apple-generated sound:
FACE   SOUND

Synthesized speech:
TALK   SPELLTALK   SPELLTALK-2

Language boards:
TALKBOARD-3X3   TALKBOARD-4X4   TALKBOARD-2X2
TALKBOARD-BINARY   TALKBOARD-GRAPHICS   TALKBOARD-MG

Echo applications:
READTALK   WORDTALK   MORSE CODE
SWITCH/SCREEN CONNECT

Chapter 5

Single switch scanning:
SCANCOLOR   SCAN DRAW   LOWRES MATCH
LOWRES NUMBER MATCH   LOWRES SCAN

Scanning techniques:
STIMULUS SCANNING   OPENSCAN   OPENSCAN-2

Chapter 6

High-resolution graphics:
HIRES COLOR   DOT-TO-DOT

High-resolution shape construction:
HIRES SHAPES   DISCRIM   SHAPE TABLE
SHAPE TABLE REVIEW   SHAPE TABLE MAKER

Large font graphics:
LASCII   DISPLAY FONT   SWITCH FONT   FONT MATCH
FONT WRITER   FONT FEEDBACK   RASCII DISPLAY FONT
RASCII SWITCH FONT   CUED LANGUAGE

High-resolution single switch applications:
HIRES GRAPHICS   HIRES AUTHOR   HIRES PAGE SWITCH

Chapter 7

Single switch math applications:
NUMBER COUNT   NUMBER MATCH   NUMBER CONCEPT
INTERVAL INPUT   MATH PROBLEMS   OPEN-ENDED SCAN

Large font math applications:
FONT COUNT   FONT MATH PROBLEMS

Developing single switch math systems:
MATHSCAN SWITCH CALCULATOR

Chapter 8

Single switch reading:
LETTER RECOGNITION CODING READ

Large font reading applications:
SCAN LETTER FONT SCAN WORD FONT

Single switch reading techniques:
SENTENCE CLOZE COLUMN MATCHING

Single switch text reading:
SWITCH READER TEXT FILE MAKER TEXT FILE READER
SWITCH READER-2 READ-2 DOS CONTROL READ AUTHOR
RESULTS READER

Program Disk

For a copy of the Program Disk containing all the programs and utilities described in this manual, please send $5 to cover the cost of mailing to

Edward Burns
School of Education and Human Development
State University of New York at Binghamton
Binghamton, New York 13902-6000.

Chapter 1

Single Switch Fundamentals

Using Single Switch Software

The single switch applications presented in this manual can be used with individual's requiring an adaptive input system, and students having a variety of learning, developmental and sensory (e.g., visual) disabilities. The programs and concepts cover a wide range of topics and difficulty levels. The applications are intended to be used and modified to demonstrate single switch applications, and as a program source for meeting the learning needs of person's requiring an adaptive input system.

Most of the programs can be used with little or no knowledge of BASIC programming. However, for those interested in being able to modify and individualize single switch programs, BASIC programming concepts are considered in the context of a single switch software environment. Many of these concepts are straightforward, while others are only of use to those with an interest in certain technical aspects of single switch BASIC programming. Overall, an attempt has been made to demonstrate how a variety of programming concepts and strategies can best meet the single switch software needs of a population which is quite varied with respect to age, ability and adaptive needs.

Necessary Equipment

In order to use this manual an Apple computer and at least one disk drive is needed. A printer is optional but is valuable for listing BASIC single switch programs. An initialized disk is also needed for saving the various program applications discussed in the manual.

Concerning input devices, a variety of switches can be used to provide single switch input such as tread, leaf, puff and mercury switches. In order to really use single switch software, some type of device for entering single switch responses is needed. As will be shown in the manual, a joystick or the keyboard can also be used to provide switch input.

Program Disk

Whenever using and/or modifying single switch BASIC programs, there is usually a need to save a copy of each program entered or modified on disk. To save a program on disk an initialized disk is required; that is, a disk must be used that has been prepared for use with the Apple **D**isk **O**perating **S**ystem (**DOS**). If a copy of the program disk containing the major programs described in this manual is being used, this disk can be used to initialize blank disks and to save program applications that have been changed or modified.

Single Switch Connections

If a single switch is being used, the device is connected to the nine-pin game port located on the back of the Apple. Many switches require an interface box. The switch is connected to the interface box and the interface box is connected to the game port. An interface box is used to read several different types of single switch input. If an Apple II+ is being used, the different types of single switch input. If an Apple II+ is being used, the single switch is connected to a 16-pin input/output connector located on the inside of the II+.

If the switch device being used is connected either to the nine-pin game port or to an Adaptive Firmware Card (**AFC**) I/O box, make the switch connection before the power is turned ON. Because the programs in this manual are designed to read switch input, the NORMAL setup from the AFC extended menu should be selected.

For many applications, a joystick or joystick button can be used to provide input similar to that of a single switch device. If a switch or joystick is not available, a specific keyboard key (e.g., the space bar) can be programmed to sense a switch response.

Booting Up DOS

To "boot" DOS means to "pull up" DOS (as in pulling up a pair of boots by the bootstraps). To boot DOS an initialized disk containing DOS must be used. The program disk that accompanies this manual can be used to boot DOS and to initialize other disks.

There are three methods for booting DOS once an initialized disk containing DOS is in drive #1:

1) When the Apple power is switched to ON, the system is automatically booted. To re-boot the system after the power has been switched OFF, wait 20 seconds or so before turning the power back to ON. To boot DOS using the program disk, insert the disk in drive #1, turn the power to ON, and after a bit of buzzing, DOS is booted and the message contained in the HELLO or greeting program appears, beginning with the following:

A Manual for Single Switch and

Adaptive Software Programming

2) To re-boot the system while in an application, press the following combination:

Control + open-Apple Key + Reset

For IIGS users, the Reset key is designated by a triangle and is located at the top of the keyboard.

3) If in BASIC programming mode (as indicated by the **]** prompt), either of the following re-boots the system:

```
PR#6    (press RETURN)
IN#6    (press RETURN)
```

Initializing Disks

There are two versions of DOS which can be used with the Apple. One frequently used version is called **DOS 3.3**. A more advanced version of DOS is also available and is called **ProDOS** (which stands for Professional Disk

Operating System). The program disk for the collection of single switch applications in this manual uses DOS 3.3.

BASIC programming mode is signified by the blinking cursor immediately to the right of the bracket:

```
]■
```

In order to initialize a DOS 3.3 disk, insert the program disk or the DOS 3.3 System Master disk in dive #1 (not #2) and boot the system (e.g., turn the Apple "on"). If the program disk or system master disk is not available, a disk already formatted in DOS 3.3 can be used to boot up the system.

The first program on the program disk contains the greeting or HELLO program. After the disk has been entered and the system booted, the HELLO program can be listed by entering LIST, and a complete listing of the HELLO program will appear:

To initialize a disk, enter the "greeting" or HELLO program. Each time the system is booted, the greeting program is automatically run. There are many ways in which a greeting program can be used in conjunction with single switch software. The HELLO program on the program disk provides several ideas for using different BASIC statements for introducing a disk by a series of screen displays when the HELLO program is first run.

Unless you want to use the program disk HELLO program as the greeting program for a new disk, clear this program from memory by entering NEW and then pressing the RETURN key:

```
]NEW (Press RETURN)
```

Next, remove the disk containing DOS from the disk drive and insert a blank disk. To actually initialize the disk the Apple must be in BASIC mode. This means that the Apple has been booted and is ready to accept and execute (i.e., run) a BASIC program.

The following program is very simple in that three things happen when the system is booted and this program is run: 1) the screen is cleared by the HOME instruction in line 10, line 20 prints a name, and line 30 prints the date listed.

```
10 HOME
20 PRINT "Ed Burns"
30 PRINT May 1, 1992"
```

To initialize a disk using a very simple greeting program, enter NEW to erase the current program in memory and then enter these lines with the appropriate information:

```
NEW
10 HOME
20 PRINT "name"
30 PRINT "month, day, year"
```

To initialize the blank disk in drive #1, enter INIT and press RETURN. Be certain when initializing a disk that the disk is blank or is a disk containing programs no longer wanted. When a disk is initialized, the previous disk contents of the disk are erased.

```
INIT HELLO
```

The red light on the disk drive is appears, and after a bit of disk switch BASIC programs.

When a program is entered or modified, the program is often saved on disk for use at a later time. To save a program on disk, the SAVE command is used along with a file name. For example, to save a program called DEMO, enter the following and then press RETURN:

```
SAVE DEMO
```

If a program already exists with the file name used to save the program, the old file is destroyed and replaced by the new file. As a result, different programs or variations of the same program should be saved using different file names. To save a modified version of the SWITCH DEMO program, the following types of names could be used: SWITCH DEMO-1, SWITCH-1, SWITCH MODIFIED.

Although a file can contain whatever characters are deemed appropriate, the file name must begin with a letter, cannot contain a comma, and is 30 characters or less. If possible, keep file names to 10 characters or less. The following are acceptable file names: SW, SWITCH #1, SWITCH MODIFICATION. An example of an unacceptable file name is 3SWITCH or *SWITCH.

The DOS Catalog

Each time a program is saved on disk, the file name is added to the disk catalog. To display the catalog, enter CATALOG and press return. For ProDOS user's, either CAT (which provides abbreviated catalog listings) or CATALOG can be used to list disk programs.

```
CATALOG
```

When the disk catalog is displayed, several types of catalog entries are displayed. The following catalog list is for the program disk containing the primary programs described in this manual.

```
DISK VOLUME 254

*A 005 HELLO
 A 002 DEMO
 A 003 ROCKET
 A 002 SWITCH COLOR
 A 003 TEST
 A 004 INPUT CHECK
 A 003 HELLO-U
 A 004 DISK MENU
 A 003 LOWRES SAMPLE
 A 003 LOWRES-SG
 A 004 NILT
 A 005 SCAN TRACKING
 A 003 ARCADE
 A 003 JOYSTICK
 A 003 JOYSTICK FEEDBACK
 A 002 TOUCHWINDOW
 A 006 LOWRES AUTHOR
 A 002 LOWRES SCREEN
 B 006 YES
 A 002 MEMORY MOVE
 A 003 LOWRES SCREEN-2
 A 004 FACE
 A 003 SOUND
 B 004 TEXTALKER
```

```
B 048 TT.OBJ
A 004 TALK
A 006 SPELLTALK
A 007 SPELLTALK-2
A 006 TALKBOARD-3X3
A 006 TALKBOARD-4X4
A 005 TALKBOARD-2X2
A 005 TALKBOARD-BINARY
A 005 TALKBOARD-GRAPHICS
A 006 TALKBOARD-MG
A 008 READTALK
A 006 WORDTALK
A 005 MORSE CODE
A 009 SWITCH/SCREEN CONNECT
A 004 SCANCOLOR
A 004 SCAN DRAW
A 003 LOWRES MATCH
A 004 LOWRES NUMBER MATCH
A 005 LOWRES SCAN
A 005 STIMULUS SCANNING
A 006 OPENSCAN
A 004 OPENSCAN-2
A 003 HIRES COLOR
A 003 DOT-TO-DOT
A 003 HIRES SHAPES
A 005 DISCRIM
A 003 SHAPE TABLE
A 004 SHAPE TABLE REVIEW
A 004 SHAPE TABLE MAKER
B 015 LASCII
A 003 DISPLAY FONT
A 005 SWITCH MATCH
A 003 FONT MATCH
A 005 FONT WRITER
A 007 FONT FEEDBACK
B 010 RASCII
A 003 RASCII DISPLAY FONT
A 005 RASCII SWITCH FONT
A 006 CUED LANGUAGE
A 004 HIRES GRAPHICS
A 005 HIRES AUTHOR
A 003 HIRES PAGE SWITCH
A 004 NUMBER COUNT
A 005 NUMBER MATCH
A 006 NUMBER CONCEPT
A 005 INTERVAL INPUT
A 004 MATH PROBLEMS
A 004 OPEN-ENDED SCAN
A 003 FONT COUNT
A 006 FONT MATH PROBLEMS
A 009 MATHSCAN
A 010 SWITCH CALCULATOR
A 003 LETTER RECOGNITION
A 005 CODING
A 004 READ
A 008 SCAN LETTER FONT
A 011 SCAN WORD FONT
A 007 SENTENCE CLOZE
A 009 COLUMN MATCHING
A 006 SWITCH READER
A 003 TEXT FILE MAKER
A 002 TEXT FILE READER
```

```
A 006 SWITCH READER-2
A 007 READ-2
A 004 DOS CONTROL
A 007 READ AUTHOR
A 003 RESULTS READER
T 002 CONTROL.SW
T 002 WORDS
T 002 RESULTS
```

After saving a program on disk, or to see what is on the disk, use the CATALOG command. The information in the catalog indicates the type of files on the disk, the relative size of each file, and the name of each file on disk. Unless otherwise changed (and it usually isn't), the volume number is always 254. Each catalog entry provides three important pieces of information: file type, size, and name.

```
A        002        SWITCH
```

The **A** file type shown above signifies that the file is an Applesoft or BASIC file. Other file types include Binary (**B**), Integer (**I**) and Text (**T**) files. Only files designated by the letter **A** can be modified using Applesoft BASIC.

The number to the right of the file type indicates the size of the file in terms of sectors. The SWITCH file shown in the above CATALOG entry is comprised of two sectors. One sector can store what amounts to 256 characters (e.g., letters, numbers, symbols). In all, each disk can accommodate up to 560 sectors. For each disk, DOS requires 48 sectors and the disk directory another 16 sectors. Thus, of the 560 sectors on a disk, 496 are available for use to store programs and files.

The program disk comprising the programs in this manual use 492 sectors, leaving only 4 sectors free. As a result, if you modify a program, or develop an application for a specific individual, you will want to do so on a separate initialized disk. You might also want to save programs on separate disks by the type of single switch task involved (e..g, a disk for reading programs, another for language boards), or you might want a separate disk for each student, or for several student's having similar single switch needs. Regardless of how you save various program applications, you will want at least one initialized disk when using the manual and program disk.

The size of a program can effect the speed of the program to process statements, and the extent to which graphics can be used. What is a large or small program? The smallest BASIC program requires 2 sectors; programs less then 10 sectors are in the small category; many programs tend to be in the 10 to 30 sector range; and programs comprised of more than 30 sectors can be considered large.

Using ProDOS Programs

If you attempt to catalog a disk and the message DISK VOLUME 001 appears, followed by a bit of unsuccessful disk buzzing and an I/O ERROR (i.e., input/output error), you are probably trying to access a ProDOS disk using DOS 3.3 (or vice versa). This will not work. In order to use a BASIC program, the program must be compatible with the disk operating system in use. Thus, you will need to convert the ProDOS program to DOS 3,3, or the DOS 3.3 program to ProDOS.

To transfer a single switch program, application or disk file from a ProDOS disk to a DOS 3.3 disk or vice versa, first boot the ProDOS User's Disk. Next, use the DOS <-> PRODOS CONVERSION option to access the CONVERT

Disk. Next, use the DOS <-> PRODOS CONVERSION option to access the CONVERT menu. The first two options from this menu are used to set the direction of transfer and the DOS 3.3 slot drive.
When you first access the CONVERT menu, the program is set to convert a DOS 3.3 program in drive 2 to a ProDOS program in drive 1:

Direction: DOS 3.3 S6,D2 ---> ProDOS

To convert a ProDOS file to DOS 3.3, reverse the direction of transfer. Remember that to transfer files you must transfer from a DOS 3.3 formatted disk to a ProDOS formatted disk. Consult the ProDOS User's Manual for a detailed discussion of ProDOS/DOS 3.3 file conversion.

For the most part converted program will work quite well. If a conversion difficulty does occur, it will probably involve a fairly complex program which accesses the disk directory or uses random-access files (which will not convert).

The advantages of DOS 3.3 is ease of use. Insert a formatted DOS 3.3 disk and your in business. The advantages of ProDOS include faster disk access and use with all types of disks (i.e., hard and floppy). Unless you have special programming needs, single switch programs using DOS 3.3 will work just fine.

Renaming and Deleting Files

File names can be re-named by using the RENAME command. If a file is named DEMO and the name DEMONSTRATION is wanted, enter the following:

RENAME DEMO,DEMONSTRATION

However, be careful not to rename a file using a name already contained in the catalog. If this occurs, a problem might result when attempting to access files with duplicate names.

Files can be deleted from the catalog by means of the DELETE command.

DELETE X

Once a file has been deleted, it is gone (unless a program is available which undeletes files) so be careful when deleting files. To see what deleted files are on disk, enter the following and press RETURN.

POKE 44505,234: POKE 44506,234

Accidental file deletions can be prevented by locking files using LOCK, followed by the file name. Locked files (see the catalog shown above) are designated by an asterisk (*). Locked files can be unlocked by using **UNLOCK**.

LOCK HELLO

Locked files are shown with an asterisk (*) when the disk is cataloged:

***A 003 HELLO**

The DOS 3.3 System Master

In addition to being used to boot or startup the system, the DOS 3.3 System Master disk contains a program called **COPYA** that can be used to duplicate or copy disks. If a two drive system is being used, run the COPYA program and set the input values to the default values by pressing the RETURN key. The screen shows that the original disk is in SLOT 6 and DRIVE 1, and the duplicate disk (the blank disk) is in SLOT 6 and DRIVE 2. In almost all cases, the slot #6 (which can be seen when the cover of the Apple is removed) contains the controller card for the disk.

Before inserting the original disk into drive 1, put a piece of tape over the notch on the disk. This prevents the accidental destruction of the original disk. Now put the original disk in drive 1 and the duplicate in drive 2, press RETURN, and a copy of the original is quickly made.

BASIC Fundamentals

The word **BASIC** is an acronym for **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode. BASIC consists of a series of instructions that are given to the computer to perform a series of tasks. As already discussed, the Apple system is ready for programming when the bracket and flashing cursor appears.

If an asterisk (*****) appears instead of a **]**, the system monitor has been entered. This is a very technical area of the Apple hardware which controls and supervises how the Apple reads, processes and displays information. If a greater sign appears (**>**), the programming mode is Integer BASIC. All the programs discussed in this manual are written in **Applesoft BASIC** which is signified by the right-hand square bracket and blinking cursor. If an * or > appears, use Control+C or the Control+reset combination to return to Applesoft BASIC or enter **FP** (which stands for floating point) and press RETURN.

One frequently BASIC instruction is the HOME command. Each time this instruction is given, the screen is cleared and the cursor is positioned in the upper left-hand corner of the screen. Enter the word HOME after the bracket and press RETURN:

```
]HOME
```

Display Output

Basic can be used to print strings (i.e., a series of letters or keyboard characters) and answers to a variety of problems. Enter each of the following PRINT instructions, press RETURN at the end of each instruction, and note the result:

```
PRINT "SINGLE SWITCH SOFTWARE"

PRINT "(PRESS SWITCH TO CONTINUE)"

PRINT "CORRECT!"

PRINT 6+8

PRINT "6+8"

PRINT 8*4

PRINT 40/9
```

The * symbol in BASIC signifies multiplication and is used in many single switch programs. In terms of operation precedence, multiplication and division precede addition and subtraction. In the following examples, the **?** is used in place of PRINT (actually enter the ? symbol in place of PRINT):

```
?"SPELLTALK"
SPELLTALK

?50+4*10
90
```

The PRINT statement is a much used instruction in BASIC and can perform many tasks ranging from displaying output, performing calculations and controlling how output is displayed. The following skips four screen lines using a series of PRINT statements:

```
PRINT
PRINT
PRINT
PRINT
```

Programming statements can also be combined or "compacted" on a single lines by means of a colon:

```
PRINT: PRINT: PRINT: PRINT
```

The advantages of compacting statements include the ability to enter multiple statements quickly, faster programming speed, and a small reduction in program memory (i.e., two bytes per line number). The disadvantages include difficulties in understanding, re-entering, de-bugging compacted lines. As a general rule, compacting is useful when the statements compacted are not extremely complex, when there is a need to increase programming speed, or to reduce the size of the program. In any case, with or without compacting, a BASIC line cannot exceed 239 characters.

The following uses two compacted PRINT statements to print the string **SCAN SPEED =** on one line, and the value 10 on the next line:

```
PRINT "SCAN SPEED = ": PRINT 10
SCAN SPEED =
10
```

Punctuation in the print instruction can be used to display screen information in separate fields. Enter the following statement first with a comma separating the words in parentheses, then use a semicolon:

```
PRINT "SINGLE","SWITCH"
SINGLE          SWITCH

PRINT "SINGLE";"SWITCH"
SINGLESWITCH
```

A TAB can be used in a PRINT statement to begin printing in a specific screen column:

```
PRINT TAB(35);"SINGLE SWITCH"
                                  SINGLE SWITCH
PRINT TAB(5)"SINGLE"TAB(20)"SWITCH"
    SINGLE               SWITCH
```

Variables

Virtually every single switch program written in BASIC uses one or more variables. A variable is simply a name which begins with an alphabetic character that represents a quantity that can vary. In BASIC there are two types of variables: arithmetic variables and string variables. The variable **ST** might be used to designate the amount of "scan time," while **N$** is used to indicate the student's name. Enter the following for an example of each:

```
ST = 9
PRINT ST
9

N$ = "Elizabeth"
PRINT N$
Elizabeth
```

Or the name used in variable N$ can be a part of a feedback routine following a correct answer:

```
PRINT "CORRECT, "N$"!"
```

But if you try and mix variables, an error will occur:

```
ST = "SINGLE SWITCH"
?TYPE MISMATCH ERROR
```

Arithmetic variables such as ST above can be set to whatever arithmetic statement or expression follows the equal sign; string variables are set to whatever is contained between the quotation marks (or the string of characters).

Just as it is often necessary to set the contents of a variable, there are occasions when it is also necessary to clear a variable. Arithmetic variables are cleared by setting the variable to 0, while string variables are cleared by setting the variable to empty as indicated by two quotation marks side-by-side.

```
ST = 0
W$ = ""
```

To clear all the variables in a program, a CLEAR statement can be used.

```
W$="HELLO"
PRINT W$
CLEAR
PRINT W$
```

At one time BASIC variables were set using LET:

```
LET N = 20
PRINT N
20
```

Although LET might be used in some programs, this instruction is not necessary and is not used in the programs described in this manual.

Variable Names

Regardless of whether a variable is used to store an arithmetic value or a string, the variable name must begin with an alphabetic character, and the name must be 238 characters or less. But using a variable with a 238 character name is not recommended. Actually, only the first two letters are used to distinguish variables. Thus, if the variable **SCAN TIME** is used to signify Scan Time, **SC** would be the actual variable containing the scan time value. Try the following:

```
SCAN TIME = 50
PRINT SCAN TIME
PRINT SC
PRINT SCHOOL
```

In each of the above examples, the value 50 is displayed.

On a few occasions, a **%** might appear immediately after a variable. This indicates an integer variable and is often used to save space in large programs when using arrays In any case, variables such as **T** (arithmetic), **T$** (string) and **T%** (integer) all represent different types of variables.

Finally, concerning variables, certain variable names cannot be used. The following will result in a syntax error because PRINT is a reserved word. All words used to provide BASIC instructions are reserved and cannot be used as variable names:

```
PRINT = 4
?SYNTAX ERROR
```

Variable Input

There are several methods frequently used to input data variable information into a single switch program. The GET$ statement inputs a single character, while the INPUT statement reads a string of characters. To use either of these statements, line numbers must be specified.

```
10 GET KY$: PRINT KY$
RUN                       (Enter RUN and press RETURN)
(Press the A key)
A

10 INPUT S$: HOME: PRINT S$
RUN
?A MERCURY SWITCH CAN BE USED AS A SINGLE SWITCH.
A MERCURY SWITCH CAN BE USED AS A SINGLE SWITCH.
```

Video Mode

The characters displayed on screen can be inverted or displayed in flashing mode:

```
INVERSE
PRINT "ADAPTIVE SOFTWARE"
FLASH
PRINT "ADAPTIVE SOFTWARE"
NORMAL
```

The NORMAL command sets the screen back to the normal display mode. If the display mysteriously begins to flash or show output in inverse mode, the problem is probably due to the fact that a NORMAL statement has not been included in the program.

INVERSE and FLASH mode can also be used to generate a cursor used in scanning alternatives. A program might have three alternatives (e.g., the letters **E E** and **H**) displayed on screen, two of which are identical and one which is different. A scan or cursor is then used to highlight each alternative. If the switch is engaged while an alternative is being scanned, this alternative is treated as the person's response (i.e, the alternative that is different).

A scan or cursor can be generated by setting a string variable to a number of blank spaces, and then printing the string of blank spaces in inverse mode:

```
C$ = "   "        (The parentheses enclose three spaces)
INVERSE: PRINT C$: NORMAL
```

The scan variable can also be set to three spaces by the following:

```
SC$ = CHR$(32) + CHR$(32) + CHR$(32)
INVERSE: PRINT SC$: NORMAL
```

Switch Input

Being able to interact with a program is an essential feature of most single switch programs. The most frequently used method for sensing a switch response is to connect a switch to the nine-pin game port and then using PEEK(-16287) to determine whether the switch has been activated. Enter PEEK(-16287) as shown and then press RETURN:

```
PRINT PEEK(-16287)
32
```

The value returned should be 32 (or a value less than 128). Now enter the same statement, but press the open-Apple key (which is the same as engaging a single switch device) or hold down the switch connected to the game port while the RETURN key is pressed. The value returned is a number greater than 127:

```
PRINT PEEK(-16287)
160
```

This single instruction, being able to sense whether a switch has been activated or not, is the basis of the readiness, graphics, academic and language programs described in this guide. Of all the BASIC instructions that are used in single switch programming, PEEK(-16287) is one of the most important.

Just as PEEK(-16287) corresponds to input from the open-Apple key, PEEK(-16286) senses input from a single switch device that corresponds to the closed-Apple key. Use PEEK(-16286) as was done above to see the type of numeric values stored in this register.

Strings

The contents of string variables must often be analyzed in terms of length and individual characters. To determine the length of a string, the LEN function is used:

```
X$ = "SINGLE SWITCH SOFTWARE"
PRINT LEN (X$)
22
```

Individual string characters can be accessed by using the MID$ instruction:

```
PRINT MID$(X$,5,1)
5
PRINT MID$(X$,15,8)
SOFTWARE
```

The LEFT$ and RIGHT$ instructions can also be used to specify string segments:

```
PRINT LEFT$(X$,6)
SINGLE
PRINT RIGHT$(X$,15)
SWITCH SOFTWARE
```

BASIC has several built-in features that are sometimes used in single switch programs. The CHR$ instruction returns a keyboard character that corresponds to the ASCII code entered:

```
PRINT CHR$(65)
A
PRINT CHR$(66)
B
PRINT CHR$(67)
C
```

Each keyboard character has a corresponding CHR$ equivalent. A single switch program might refer to CHR$(27) which signifies the Escape key, to CHR$(32) which represents the spacebar, to CHR$(65) which signifies the letter A, or to CHR$(13) which indicates the RETURN key. The CHR$ function can be used to print all possible keyboard characters, upper- and lowercase characters, and all nonalphanumeric characters such as the cursor symbol ■ or CHR$(255). Try the following:

```
PRINT CHR$(83);CHR$(119);CHR$(105);CHR$(116);CHR$(99);CHR$(104)
```

The counterpart of CHR$ is the ASC function which returns the ASCII code for the string variable entered:

```
PRINT ASC("A")
65
```

Loops

The ability to perform instructions quickly is an essential feature of every computer. Enter the following line and press RETURN and the numbers 1 to 100 are displayed in sequential order:

```
FOR K = 1 TO 100: PRINT K: NEXT K
```

A variation of the above loop, without the second statement, is often used to create a pause or delay in the program:

```
FOR D = 1 TO 750: NEXT D
```

or

```
FOR L = 1 TO 1500: NEXT L
```

A delay loop might be used following screen feedback and before the screen is cleared and the next item presented.

A loop is often used to read a switch response:

```
FOR SP = 1 TO 500: PRINT PEEK(-16287): NEXT SP
```

After entering the above line and pressing RETURN, press the open-Apple key several times to see how switch input is sensed and stored in numerical form.

To read input from the closed-Apple key, enter the following:

```
FOR L = 1 TO 400: PRINT PEEK(-16286): NEXT L
```

A Loop can also be used in conjunction with variables in order to set the length of time a variable is scanned. After entering the following two lines, re-enter the lines by setting SP to 500. Try pressing the switch or open-Apple key while the loop is being executed.

```
ST = 100
FOR K = 1 TO ST: PRINT K" "PEEK(-16287): NEXT
```

In the above example, the variable K following NEXT has been omitted. This is sometimes done to increase the speed of a program, although this can make the logic of a program somewhat difficult to follow (especially if you are not aware that this programming option is possible).

Screen Position

Displaying information at specific screen locations is important in the development of all single switch programs. In Applesoft BASIC the instructions VTAB and HTAB are used to position the cursor at specific vertical and horizontal screen locations. The Apple screen consists of 24 rows so that VTAB is followed by a value from 1 to 24. To position the cursor at the 12th row and print a $ symbol, set VTAB to 12.

```
HOME
VTAB 12
PRINT "$"
VTAB 5: HTAB 10
PRINT "VERTICAL LINE 5"
```

Graphics

Graphics play an important role in the development of single switch software and the Apple has two different graphics systems: low-resolution and high resolution graphics. Enter GR to change the Apple screen to the low-resolution graphics mode. In this graphics mode there are 15 possible colors numbered 1 to 15. When low-resolution graphics is first called, the color is initially set to black.

After entering GR, set COLOR to 15 (white) and the Apple is ready to plot in low-resolution graphics. Begin by plotting a low-resolution dot in the center of the screen using the PLOT instruction:

```
GR
COLOR=15
PLOT 20,10
```

The PLOT statement can be read as "Put a low-resolution dot in column 20 and at row 10 of the low-resolution graphics screen." The format for the PLOT statement is

```
PLOT COLUMN C, ROW R
```

More low-resolution dots can be plotted by using additional PLOT statements.

```
PLOT 19,11
PLOT 21,11
```

The HLIN instruction is used to draw a straight at a specified horizontal or row screen position:

```
HLIN 0,39 AT 10
```

In words, the above instruction reads "draw a low-resolution horizontal line from column 0 to column 39 in row 10" or

"draw a **H**orizontal **LIN**e from column **0** to column **39 AT** row **10**"

Likewise, the VLIN instruction also draw a straight line but at a specified vertical or column position:

```
VLIN 12,39 AT 15
```

This instruction can be interpreted to mean "draw a low-resolution vertical line from row 12 to row 39 in column 15," or

"draw a **V**ertical **LIN**e from row **12** to row **39 AT** column **15**"

If a color monitor is being used, the color can be reset by using one of the other low-resolution color codes (which includes the codes 0 to 15). Even if a color monitor is not being used, the following results in a line that is a slightly different shade because of the difference in screen color resolution:

```
COLOR=3
HLIN 22,36 AT 7
```

As can be seen, with a little effort (and many PLOT statements) an unlimited number of images can be displayed in different shapes and colors. If using a color monitor, change the color code and try plotting several low-resolution dots.

Enter the following to see how a low-resolution screen can be enclosed in a border:

```
GR
COLOR=15
HLIN 0,39 AT 0
HLIN 0,39 AT 39
VLIN 0,39 AT 0
VLIN 0,39 AT 39
```

The second graphics system is called high-resolution graphics. This is called by entering HGR and then setting the color. While low-resolution graphics has 15 color codes, high-resolution has only seven. When calling high-resolution graphics and setting the color, be sure to use HGR and HCOLOR.

To see just how small a high-resolution dot is, enter the HPLOT statement shown below to display a dot in the center of the screen:

```
HGR
HCOLOR=7
HPLOT 135,80
```

High-resolution graphics has considerable programming flexibility when drawing lines in that a line can be drawn between any two coordinates. Enter the following:

```
HPLOT 200,10 TO 25,150
```

where the high-resolution line coordinates are designated using the following format:

```
HPLOT COLUMN,ROW TO COLUMN,ROW
```

In order to leave Apple graphics and re-enter text mode, enter TEXT or Press Control+Reset:

```
TEXT
```

PEEK's and POKE's

The Apple's memory consists of a series of registers, and each register is capable of storing a value. The contents of each register can be viewed by means of a PEEK. As already discussed, PEEK(-16287) indicates whether or not a single switch device connected to the nine-pin game port is engaged. For Apple IIe's and IIc's and IIGS's when the switch is open, the value in PEEK(-16287) is less than 128; when the switch is engaged (or the open-Apple key is pressed), the value in memory location PEEK(-16287) changes to a value greater than 127.

Because some software programs use either PEEK(-16287) or PEEK(-16286), an interface box is often used to connect the switch device being used to the correct PEEK. If the switch is connected to PEEK(-16287) but the software is using PEEK(-16286), nothing happens when the switch is engaged. As it is, PEEK(-16287) is used in most single switch programs and is used throughout this manual.

Another frequently used PEEK is PEEK(-16384) which is used to read keyboard input. Enter the following and press RETURN:

```
FOR K = 1 TO 500: PRINT PEEK(-16384): NEXT
```

At first the value 13 is displayed. However, when a keyboard key is pressed. The value displayed is greater than 127. When the **A** key is pressed, the value 193 appears; and when the Esc key is pressed, the value 155 appears.

PEEK's are also used for special functions. For example, PEEK(-16336) produces a click, and a series of these click's produces a rather grating buzzing sound. This is discussed in Chapter 4, but for know enter the following and press RETURN to sample the Apple PEEK(-16336) "click":

```
FOR K = 1 TO 50: X=PEEK(-16336): NEXT
```

A very important use of PEEK's is to examine the contents of a specific memory register. For example, memory location 78 is constantly changing so that this memory location is often used to generate random numbers (a technique which is described in detail in later chapters). Enter the following to see how location 78 changes:

```
FOR K = 1 TO 500: PRINT PEEK(78)" ";: NEXT
```

PEEK's can also be used to supply a wealth of information pertaining to the screen and general operation of a program. PEEK(37) signifies the vertical position of the cursor and PEEK(103) + PEEK(104) * 256 the start of an BASIC program (which is usually 2049).

For certain programming activities, the value stored in a register must be changed. This is accomplished by using a POKE. First enter HOME to clear the screen, then POKE the value 193 into address 1338:

```
HOME
POKE 1338,193
```

The register 1338 is part of the screen display which includes registers 1024 to 2048. The value 193 corresponds to the letter **A**. Now try this POKE:

```
POKE 1338,65
```

This displays the letter A at location 1338 but in FLASH mode.

POKE's can be used in various ways to reset standard settings. Enter the following and the message is printed in inverse mode. The second poke resets the text output format to normal mode:

```
POKE 50,63
PRINT "SINGLE SWITCH SCAN SPEED"
POKE 50,255
```

Commonly used POKE's include POKE-16368,0 which is used immediately following PEEK(-16384) to set the strobe back to a value less than 128; POKE 216,0 which allows normal error messages following an ONERR GOTO; POKE 34,X which sets the top monitor display margin; and POKE 35,X which sets the bottom monitor display margin.

CALL Statements

In addition to PEEK's and POKE's, a program might contain one or more CALL statements. This is a machine language subroutine that begins at the location specified. The HOME statement previously used has a CALL counterpart which clears the screen and moves the cursor to the upper left-hand corner.:

```
CALL -936
```

And CALL -1184 clears the screen and prints **Apple][** at the top of the screen.

The following provides a sampling of various CALL statements which can appear in a single switch application:

CALL -198	(Ring bell)
CALL -868	(Clear line from cursor to end of line)
CALL -922	(Move cursor down one line)
CALL -958	(Clear cursor from cursor to bottom of screen)
CALL -998	(Move cursor up one line)
CALL -1370	(Boot system)
CALL -3100	(Display the high-resolution screen)

This last CALL provides yet another method for booting the Apple system. For a quick sampling of CALL statements try and determine what happens before the following is entered and RETURN pressed:

```
CALL -936: FOR K = 1 TO 10: CALL -198: CALL -922: NEXT
```

Single Switch BASIC Programs

For the most part , most of the example thus far given has used **immediate**

execution mode of BASIC; that is, the statement is executed as soon as the RETURN key is pressed. In order to realize the full capability of BASIC, and to save and run programs using DOS, **deferred execution** must be used. Deferred execution is used to write a series of statements (i.e., a program), and then to run the collection of statements as a single BASIC program.

The following is an example of a single statement program:

```
10 PRINT PEEK(-16287): IF PEEK(-16287) < 128 THEN 10
```

To "run" or "execute" the program, enter the very important and often used RUN command and press RETURN.

```
]RUN

RUN
32
32
32
.
.
.
32
```

PEEK(-16287) reads the register used to sense single switch input. The program terminates when the open-Apple or single switch device is engaged. The advantages of a program are threefold: 1) The program can be used repeatedly by simply entering the RUN command; 2) specific program lines can be modified; and 3) the program can be stored on disk for use at a later to time.

BASIC programs are always executed or run beginning with the lowest statement number and proceeding upward, unless the program branches to another statement within the program. The program statements in this manual have been written in increments of 10. This was done so that additional lines could be added to programs if necessary. The line increments could have just as well been 1 or 100 and the programs would function just the same.

The following short program illustrates how BASIC instructions are processed in a single switch program. When the ready symbol] appears to the left side of the screen, the Apple is ready for a BASIC instruction. To enter the first statement, enter **10 HOME** after the prompt and then press RETURN to signify the end of a single BASIC program line:

```
]10 HOME
```

Immediately after a line has been entered, the bracket appears signifying that the Apple is ready for the next BASIC instruction. Now enter the following program, and be sure to press RETURN at the end of each statement:

```
10 HOME
20 IF PEEK(-16287) >127 THEN 40
30 GOTO 10
40 PRINT "CLOSED SWITCH"
50 GOTO 20
```

The HOME instruction in line 10 clears the screen and positions the cursor to the upper left-hand corner of the screen. As discussed above, the really important instruction is contained in line 20. Every fraction of a second the Apple checks to see whether a switch connected to the nine-pin game port has been activated. If a switch has been activated or engaged, the value in computer location PEEK(-16287) is set to a value greater than 127.

When PEEK(-16287) is greater than 127 (or when the open-Apple key is pressed), the program branches to line 40 and the string CLOSED SWITCH is displayed. If the value in location -16287 is 127 or less, control is "looped" back to line 10.

In order to modify the program, the program execution must be interrupted. The next section describes how to accomplish this, depending on the Apple system being used.

Interrupting a Program

Being able to "interrupt" a program while a program is running is important in order to stop the program for one reason or another such as running another program, or to make program modifications. One method for interrupting a program is to press Control key and the Reset key at the same time. For Apple II+ users, press reset. For IIGS users, use the Control+Reset+open-Apple combination.

Interrupt Sequence	**Apple Model**
Reset	**II+**
Control+Reset	**IIc,IIe**
Control+C	**II+,IIe,IIc,IIgs**
Control+Reset+open-Apple	**IIgs**

Programs can also be interrupted by pressing Control and the C key at the same time. If Control+C is pressed and nothing happens, press the Return key immediately after pressing **Control+C**. Although Control+C is designated as an interrupt sequence, if a program is processing an INPUT statement, RETURN must be pressed after the Control+C combination in order to complete the interruption process. For the short program shown above, Control+C results in a statement such as:

```
BREAK IN 10
```

The number 10 in the above message indicates the line number that was being executed when the Control+C was pressed.

Following a program interrupt, and if no program modifications have been made and no errors have been detected, the program can generally be re-entered using the CONT command:

```
CONT
```

Because commercial software is generally not written in BASIC, these programs cannot be interrupted and modified as is the case with most BASIC programs.

Now interrupt the program by pressing Control+C and enter LIST to list all the program instructions. Modify the program by re-entering line 40 as shown and then RUN the program again:

```
]LIST

10 HOME
20 IF  PEEK ( - 16287) > 127 THEN 40
30 GOTO 10
40 PRINT "CLOSED SWITCH"
50 GOTO 20

]40 PRINT PEEK(-16287)" ";
```

The Escape key is often used to exit a program. This feature is added to the above program by line 25. The computer scans the keyboard and returns the decimal value of each key pressed. If Esc is pressed, the value 155 is generally returned. If the keyboard strobe is reset by POKE -16368,0 before each loop iteration, the Esc value 27 is returned (see line 220 of the Input Check program):

```
25 KY = PEEK(-16384)
26 IF KY = 155 THEN 60
60 POKE-16368,0
70 END
```

Or as follows if POKE -16368,0 is used within a loop:

```
25 KY = PEEK(-16384)
26 IF KY = 27 OR KY = 155 THEN 60
```

Entering New Programs

The instruction NEW erases the current program in memory so that a new BASIC program can be entered. Clear the Apple's memory using NEW and enter the following brief program which displays the name of the key pressed.

```
NEW
10 HOME
20 GET KY$
30 PRINT KY$" "ASC(KY$)
40 IF KY$ = "Q" THEN 60
50 GOTO 20
60 END
```

Although the above program is comprised of only six lines, it can be used to test the Apple keyboard keys. When run, the character and ASCII value associated with each keyboard key is displayed when the key is pressed (with the exception of one key).

The following program illustrates the use of a horizontal scan which can be used to highlight a series of alternatives in a single switch program:

```
NEW
10 HOME
20 E$ = "      "
30 H = H + 10
40 IF H > 30 THEN H + 10
50 VTAB 10
60 HTAB H
70 INVERSE
80 PRINT E$
90 FOR L = 1 TO 1000: NEXT L
100 GOTO 10
```

Internal Documentation

If a program is relatively short and simple to use, there is no need for a vast amount of external documentation. For these types of programs, several sentences might be all that is necessary to explain the purpose and how to use the program. However, as the complexity and options available within a program increase, so does the need for additional printed documentation.

In addition to external documentation, a program can be documented internally by adding REM (as in REMark) statements. A REM statement has no other purpose other than to provide information when reading the BASIC listing. For the program described above, an internal label can be added to the program by inserting two REM statements:

```
5 REM SWITCH
6 REM BY MARY JONES
```

If a program is fairly long, look for REM statements to help understand the code. After modifying a program, use a REM statement to record the date when the program modification was made:

```
30 REM
40 REM JANUARY 24, 1991
50 REM
```

If you are wondering whether a particular single switch application is in the public domain or not, a quick scan of the program listing might provide not only copyright information but also the whereabouts of the author:

```
10 REM ********************************
20 REM SINGLE SWITCH ACTIVITIES
30 REM COPYRIGHT (C) 1992 BY
40 REM EDWARD BURNS
50 REM SUNY-BINGHAMTON
60 REM BINGHAMTON, NY 13901-6000
70 REM ********************************
```

Bugs!

When programming, many factors can prevent a program from doing what it was intended to do. When this happens, when a program does the unexpected, or simply does not work, the program has a "bug." And to get rid of a bug (or bugs), is the business of "debugging."

For a quick lesson on "bugs," enter NEW to clear memory and then HOME to clear the screen. There are two main categories of bugs that are of particular concern to all persons using single switch BASIC programs: 1) statement errors, and 2) conceptual errors. Enter the following line exactly as shown (be sure to spell HOME with an N as shown):

```
30 HONE
```

and when this one line program is run the following error statement appears:

```
?SYNTAX ERROR IN 30
```

The vocabulary and syntax used to write BASIC statements must be precise. Instructions cannot be misspelled and the instructions must contain all the necessary elements. Enter the following and see what happens when the program is run:

```
30 HOME
40 VTAB
```

The first line clears the screen, but the second line results in an error statement. The reason for this is that the syntax for the VTAB statement is the instruction VTAB, followed by a row number (a value from 1 to 24). Try experimenting with different VTAB values and see what happens (try 14, 0, 24, 25 and then 1). What type of error occurs when VTAB is set to 0 or 25? For the most part, the majority of errors encountered involve BASIC statement errors, especially when copying program listings.

Sometimes a bug can be quite subtle. For example, there is a big difference between the letter **O** and the number **0**. Though it is true that 10+4 equals 14, 1O+4 equals **?SYNTAX ERROR**.

As opposed to statement errors, conceptual errors occur when a BASIC statement does exactly what it was specified to do. Unfortunately, this might not be what is really wanted. Suppose a switch program that is designed to read PEEK(-16286) to determine whether or not a switch has been engaged. If the open-Apple key is used to activate the switch, or a switch connected to PEEK(-16287), the program won't work. This is clearly a conceptual error relating to how switch input is being read.

When debugging, first determine exactly what the program does in relation to what the program should be doing. Next, locate the line number in the program where the error is occurring. In many cases, the line number where the error is occurring is displayed. If a line contains an error, simply re-enter the entire line.

BASIC Code Variations

If interested in a detailed discussion of the types of statements which are used to create Apple BASIC programs, definitely consult the **Applesoft BASIC Programming Reference Manual** which is published by Apple Computer. Concerning BASIC programming manuals, be aware that there are many variations of BASIC and that these BASIC language variations are not necessarily interchangeable.

Although different versions of BASIC (IBM, Apple, Commodore, etc.) are very similar, there are sufficient differences within each version that often require some (and sometimes many) changes when using one BASIC program with several different computers. As an example, HOME is used to clear the screen in Apple BASIC, but CLS performs the same function in IBM BASIC.

The following is an example of a PC single switch program which is modeled after the DEMO program described in the next chapter. When this brief program is run, an open switch is displayed on screen. When the switch is engaged or a key is pressed, the switch closes. The Esc key is used to exit the program (see line 100).

```
10 REM PCDEMO
20 REM
30 CLS
40 STRIG ON
50 X = STRIG (0)
60 LOCATE 10,20
70 PRINT "___________/ ___________"
80 IF STRIG (1) < 0 THEN 120
90 KY$ = INKEY$
100 IF KY$ = "" THEN 80
110 IF ASC(KY$) = 27 THEN 160
120 LOCATE 10,20
130 PRINT "_______________________"
140 FOR D = 1 TO 1500: NEXT D
150 GOTO 60
160 END
```

To run the above program, the following sequence is used:

```
A>BASICA
```

```
10 PCDEMO         (enter PCDEMO program)
20 REM
.
.
.
160 END
RUN               (or use the F2 function key)
```

To exit BASIC back to DOS, enter SYSTEM and then press RETURN.

There are many similarities between Apple and PC BASIC, yet each version of BASIC has certain unique characteristics. As can be seen from the above listing, there are a number of identical Applesoft and PC BASIC instructions. For the most part, REM, PRINT, FOR/NEXT loops, IF statements (line 110) and GOTO statements line 140) are virtually the same in the two forms of BASIC.

PC BASIC statements which have similar counterparts in Apple include CLS, STRIG(0), LOCATE and INKEY$. In line 30 CLS clears the screen just as HOME does in Apple BASIC. The STRIG (0) function in line 80 is similar to PEEK(-16287) in Apple BASIC, but when the switch is engaged STRIG(0) returns a value of -1.

The LOCATE statements in lines 60 and 120 sets the cursor position and takes the place of VTAB and HTAB so that **60 LOCATE 10,20** is equivalent to **60 VTAB 10: HTAB 20**. The INKEY$ function scans the keyboard and determines whether a key has been pressed. The Apple counterpart of INKEY$ is PEEK(-16384).

The STRIG ON statement is unique to PC BASIC. This command must be used in order to read switch input via the STRIG(1) instruction. As can be seen from the above listing, there are many equivalent Apple BASIC and PC BASIC statements. However, for single switch applications, you will invariable need to modify a BASIC program when converting programs. Also, if a non-IBM system is being used, you will need to use GW-BASIC or a similar version of BASIC.

Chapter 2

Single Switch Input

Demonstration Programs

The primary ingredient that underlies all single switch software is the use of an adaptive input device which sends a very simple message to the computer: SWITCH OPEN (no response) or SWITCH CLOSED (response). A variety of mechanical devices can be used to determine whether or not a response has been made. The most often used device is a simple plate or tread switch. When the switch is engaged, a circuit is closed thereby indicating a response has been made.

Single Switch DEMO Program

To illustrate how a single switch software program works, enter the DEMO program listing shown below. As mentioned in the last chapter, enter NEW to clear the programming memory area. If a program disk is being used, the DEMO program is retrieved from the disk by using either LOAD or RUN. The LOAD command retrieves the program from disk into memory but does not actually run the program. This is useful when modifying a program or to check a program listing. The RUN command retrieves the program from disk and then immediately runs or executes the program.

```
NEW
RUN DEMO
```

The switch DEMO program illustrates the general format used by many of the programs contained in this manual. Enter the program as shown and then run the program:

```
10 REM DEMO
20 REM
30 HOME
40 VTAB 12: HTAB 14
50 PRINT "_______/_______"
60 IF PEEK (-16287) > 127 THEN 90
70 IF PEEK (-16384) = 155 THEN 130
80 GOTO 60
90 VTAB 12: HTAB 14
100 PRINT "______________"
110 FOR D = 1 TO 750: NEXT D
120 GOTO 40
130 POKE -16368,0
140 END
```

When the DEMO program is run a graphic image signifying an open switch appears on the screen. If a single switch device is connected to the Apple, activate the switch and the switch displayed on screen closes. In line 60 when the switch is engaged, PEEK(-16287) is set to a value greater than 127 and the program branches to line 90. If a single switch device is not connected to the game port, the open-Apple key provides the necessary switch input. To exit the program, press the Esc key.

```
_______/ _______     (open switch)
```

Note that holding the switch down in a continuous manner results in a continuous series of screen activities (i.e., the opening and closing of the switch displayed). A method for requiring distinct switch movements (engaging and then releasing the switch) is discussed shortly.

```
________________    (closed switch)
```

Run the program by using both discrete and continuous switch responses. Exit the program by pressing the Esc key. The numerical code for the Esc is 155. In line 70, if the Esc key is pressed, the number 155 is stored in memory location -16384. When the contents of register -16384 is 155, control is branched to line 130, the keyboard is cleared by the POKE in line 130, and the program ends in line 140.

The following is a line-by-line description of how the program works. The delay loop in line 110 results in a delay of approximately one second. Changing the loop to

```
110 FOR D = 1 TO 1500: NEXT D
```

results in a delay of approximately two seconds.

10 REM DEMO	**Remark statement with name of program**
20 REM	**Remark statement used to space listing**
30 HOME	**Clears screen**
40 VTAB 12: HTAB 14	**Positions cursor to the 12th line and 14th column**
50 PRINT "______/______"	**Displays open switch on screen**
60 IF PEEK (- 16287) > 127 THEN 90	**Sends control to line 90 if open-Apple is pressed**
70 IF PEEK (- 16384) = 155 THEN 130	**Branches to line 130 if Esc is pressed**
80 GOTO 60	**Branches to line 60**
90 VTAB 12: HTAB 14	**Positions cursor to the 12th line and 14th column**
100 PRINT "______________"	**Displays closed switch**
110 FOR D = 1 TO 750: NEXT D	**Delay loop**
120 GOTO 40	**Branches to line 40**
130 POKE -16368,0	**Clears keyboard strobe**
140 END	**Ends program**

The value in line 70 used to indicate whether or not a keyboard key has been pressed is sometimes written as

```
70 IF PEEK(-16384) > 27 + 128 THEN 130
```

The ASCII value for the Escape key can be either 27 and 155. ASCII values are divided into two sets: low ASCII values and high ASCII values. The low values are 0 to 127, and the high values range from 128 to 255. The high ASCII value which corresponds to the low ASCII value can be found by adding 128 to the low value (e.g., 27 + 128). However, although 65 and 65 + 128 both represent the letter **A**, the Apple does not recognize the two as the same character...strange but true.

The END statement shown in line 140 is not required in that programs will automatically end when there are no more executable program lines. However, a program will immediately end when an END statement is encountered regardless of where that statement is located within a program.

Listing a Program

Whenever a program is entered, or modifications made, it is always a good idea to list the program in order to check for possible bugs. Substituting one letter for another, or entering a period instead of s colon can result in a program error. The entire program can be listed by entering LIST:

```
LIST
```

When the DEMO program is listed, the following should appear:

```
10  REM DEMO
20  REM
30  HOME
40  VTAB 12: HTAB 14
50  PRINT "______/______"
60  IF  PEEK ( - 16287) > 127 THEN
     90
70  IF  PEEK ( - 16384) = 155 THEN
     130
80  GOTO 60
90  VTAB 12: HTAB 14
100  PRINT "_____________"
110  FOR D = 1 TO 750: NEXT D
120  GOTO 40
130  POKE  - 16368,0
140  END
```

The list of the program might be somewhat different than how the program was entered for two reasons: First, spaces are automatically inserted between BASIC instructions. Second, the Apple uses its own set of rules for determining how long a line displayed on screen is before the line is continued on the next screen line. Re-enter line 40 with no spaces as follows:

```
40VTAB12:HTAB14
```

and then list the single line 40 by entering **LIST** followed by the line number:

```
LIST 40
```

and the line is displayed as:

```
40  VTAB 12: HTAB 14
```

To see a section of the program, or a series of program lines, the LIST command can be used to focus on specific program segments. Try the following commands to list different program segment and note what each command prints:

```
LIST
LIST-50
LIST 30-80
LIST 60-
```

As indicated by the above, LIST-50 lists the first 50 lines of the program (or lines 10, 20, 30, 40 and 50 in the above program); LIST 30-80 lists lines 30 through 80; and LIST 60- lists all lines beginning with line 60 to the last line of the program.

To stop a program listing (a useful technique for longer programs or when scanning a listing), use Control+S. Using this combination a second time resumes the listing:

CONTROL+S

To terminate a list in progress, use **Control+C**:

Printer Listing

If a printer is available, and the printer is in the usual #1 slot, enter PR#1 and then use the LIST command to "dump" the program listing to the printer:

```
PR#1
LIST
```

As discussed above, the LIST command can be used to print partial listings via the printer. If a listing has been dumped to the printer, enter PR#0 to return output to the screen rather than the printer (or use Control+Reset):

```
PR#0
```

Screen Speed

The Apple lists lines at a fairly rapid rate. The system speed can be slowed by using the SPEED command. This can be very useful as a very quick and simple method for slowing down single switch applications, or when debugging a program. The following setting substantially slows down the rate characters are displayed on the screen:

```
SPEED=125
```

The SPEED rate can very between 0 and 255. The normal SPEED rate is 255 and is automatically set when the system is first booted:

SPEED=0	(very slow)
SPEED=125	(slow)
SPEED=255	(normal)

To determine the speed setting currently in use, enter the following:

```
PRINT 256 - PEEK(241)
255
```

IIGS Users

Most software applications designed for the IIc and IIe run without difficulty using a IIGS and vice versa. However, certain machine language programs developed using the IIc and IIe can sometimes cause the computer to "hang" (no keyboard keys seem to work) and leave no other option but to re-boot.

For IIGS users an extremely important feature to consider is the IIGS control panel. After the IIGS has been booted, the control panel can be entered by pressing

open-Apple + Control + Esc

The control panel contains the following options:

Display
Sound
System speed
Clock
Keyboard
Slots
Printer Port
Modem Port
RAM Disk
Mouse Disk

Quit

If a single switch software applications is running very fast when using a IIGS, the system program speed might be set to FAST. Reset the system speed to NORMAL and all should run as expected.

Discrete Response Input

The DEMO program illustrates how a single switch device works, but as the program is now written there are several potential problems. One area of particular concern is the fact the program does not require discrete switch responses. Press the switch or hold down the open-Apple key and notice that the screen switch displayed is activated continuously as long as the switch is engaged.

Whether or not a single switch program should require discrete switch responses (i.e., the switch must be released before a second switch response can be registered) depends on the age and cognitive ability of the child or student and the nature of the program. For an older student able to use a single switch program to develop advanced reading skills or to develop mastery in a content area, the need for building into the software a routine that requires discrete switch responses is not a major concern. However, for a younger student who exhibits random keyboard behavior, a routine that prevents random or continuous switch responses might be extremely valuable.

There are several programming techniques that can be used to require distinct switch responses. One method is to evaluate the status of the switch prior to the switch input that is used to initiate a screen activity or immediately after the screen event has occurred. IF line 55 is added to the DEMO program, the program will hold at line 55 as long as the switch is closed:

```
55 IF PEEK(-16287) > 127 THEN 55
```

When line 55 is added and the program run, the program first checks to see if a the switch is closed. If the switch is activated or pressed, the statement continues to branch back to the same line. In other words, the program stalls until the switch or circuit is open.

Now delete line 55 and add line 95

```
55
95 IF PEEK(-16287) > 127 THEN 95
```

When this switch check is added after an actual switch response has been detected in line 60, the switch must be released before the screen activity occurs (i.e., the closed switch is displayed via line 100). For the first switch check, the switch must be open before a switch response can be made causing a screen activity to occur. For the second switch check, the switch is engaged and then must be released before the activity occurs.

A third switch check can be inserted in line 115:

```
95
115 IF PEEK(-16287) > 127 THEN 115
```

When the program is run with this check in place, a switch response causes the screen switch to close, but the switch must be released before the screen switch appears in the open position.

Yet another method for requiring discrete switch responses is to add a routine that evaluates the switch immediately after a switch response has been made. The program is held in this routine until the switch has been released. To illustrate, make these modifications:

```
114 FOR L = 1 TO 50
115 IF PEEK(-16287) > 127 THEN 114
116 NEXT L
```

This routine creates a loop so that immediately after a switch response is made, the status of the switch is evaluated the number of times specified by the second delimiting value in the loop (value 50 in line 114). For the above loop, the switch is evaluated 50 times. If the switch is engaged while in the loop, the loop begins anew with a new cycle of 50 switch evaluations. As long as the switch is engaged, the program does not leave this "holding" routine. If the switch is open for 50 iterations, the program moves to the next program component.

The advantage of a loop rather than a single evaluation is that not only must the switch be open after a response, but the switch must be open for a specified period of time. Thus, hitting the switch quickly and repeatedly does not cause the program to leave the delay routine. Experiment with the above delay routine with different types of switch responses. Also, change line 114 so that the loop contains a smaller or larger number of iterations:

```
114 FOR L = 1 TO 25     (fewer switch loop iterations)

114 FOR L = 1 TO 100    (more switch loop iterations)
```

For very young children, or students unfamiliar with computers, requiring discrete switch responses might not be necessary. Indeed, for some students, any type of response might be encouraged whether it be discrete, continuous or completely random! However, as the child or student progresses, an attempt should be made to develop the ability to provide, if possible, discrete switch responses.

Sometimes a variable is used to temporarily store the contents of the

BEST COPY AVAILABLE

switch register. In the following example, the contents of PEEK(-16287) is stored in variable X and then X is evaluated.

```
60 X = PEEK(-16287)
65 IF X > 127 THEN 90
```

Lines 60 and 65 could also be compacted using one line number:

```
60 X = PEEK(-16287): IF X > 127 THEN 90
```

Using a variable to evaluate a response is frequently used with PEEK(-16384) when determining whether or not a keyboard key has been pressed. In the DEMO program the variable KY can be set to the value in PEEK(-16384). If KY is greater than 127, the program branches to line 85, the keyboard strobe is reset so that the next keyboard response can be read, and the value in KY is then checked for an Esc response in line 86:

```
70 KY = PEEK(-16384): IF KY > 127 THEN 85
85 POKE -16368,0
86 IF KY = 155 THEN 130
```

If KY is equal to 155 (Escape), the program ends. However, all other keyboard responses are treated as switch responses by the program. This modification allows any keyboard key (other than the Esc key) to provide single switch input. As a result, simply being able to hit any keyboard key serves as a single switch response.

```
10 REM DEMO
20 REM
30 HOME
40 VTAB 12: HTAB 14
50 PRINT "______/______"
60 IF PEEK(-16287) > 127 THEN 90
70 KY = PEEK ( -16384): IF KY > 127 THEN 85
80 GOTO 60
85 POKE -16368,0
86 IF KY = 155 THEN 140
90 VTAB 12: HTAB 14
100 PRINT "_____________"
110 FOR L = 1 TO 750: NEXT L
120 GOTO 40
130 POKE -16368,0
140 END
```

To understand the function of POKE -16368 (line 85) which appears in many single switch program, change line 85 as follows:

```
85 PRINT KY
```

Now run the program and press a key. The decimal corresponding to the key is displayed, but the switch opens and closes continuously! The reason for this is that PEEK(-16384) is set to a value greater than 127 when a key is pressed, but PEEK(-16368) is not used to reset PEEK(-16384) back to a value equal or less than 127.

When looking for the switch routine in a listing, memory location PEEK(-16287) or sometimes PEEK(-16286) generally indicates the segment of the program that is used to sense the switch response. Nonetheless, determining just how the response is being read is not identical for all programs. As an example, in the following example variable X is set to 1 if the value in PEEK(-16287) is greater than 127. In the next statement X is evaluated, and if it is equal to 1, this signifies that the switch has been engaged. In other words if PEEK(-16287) is greater than 127 (i.e., the switch is engaged), X is set to 1:

```
60 X = PEEK(-16287) > 127    (If PEEK(-16287) is greater than
65 IF X = 1 THEN 90           127 THEN X = 1)
```

Sample Routines

Variations of the DEMO program can be created to provide a very simple cause/effect switch program. The program listed below displays a rocket moving upward each time the switch is engaged. Also note how a graphics effect can be created by using simple print statements. Line 100 could also be entered as

```
100 HTAB 18: PRINT "*";SPC(3);"*"
```

or as

```
100 HTAB 18: PRINT "*";CHR$(32);CHR$(32);CHR$(32);"*"
```

```
10 REM ROCKET
20 REM
30 HOME
40 IF PEEK(-16287) > 127 THEN 40
50 IF PEEK(-16287) > 127 THEN 80
60 IF PEEK(-16384) = 155 THEN 240
70 GOTO 50
80 VTAB 22: HTAB 20: PRINT "*"
90 HTAB 19: PRINT "* *"
100 HTAB 18: PRINT "*   *"
110 HTAB 17: PRINT "*******"
120 FOR J = 1 TO 7
130 HTAB 17: PRINT "*     *"
140 NEXT J
150 HTAB 17: PRINT "*******"
160 PRINT
170 HTAB 18: PRINT "*****"
180 HTAB 19: PRINT "***"
190 HTAB 20: PRINT "*"
200 FOR L = 1 TO 24: PRINT
210 FOR D = 1 TO 250: NEXT D
220 NEXT L
230 GOTO 40
240 POKE -16368,0
250 END
```

This program illustrates the shell of many cause and effect single switch programs. For each distinct switch response, control is sent to line 80 where the screen event routine begins. Following the event, control is re-directed back to line 40 and another switch response is read. If the Escape key is pressed while in the switch loop (line 60), the keyboard strobe is cleared in line 240 and the programs ends (line 250).

```
10 REM
20 REM
30 HOME
40 IF PEEK(-16287) > 127 THEN 40
50 IF PEEK(-16287) > 127 THEN 80
60 IF PEEK(-16384) = 155 THEN 240
70 GOTO 50
80
```

```
.
.
.
230 GOTO 40
240 POKE -16368,0
250 END
```

Instead of having a rocket blast off, the alphabet could be listed, the screen color changed, or a graphic image displayed each time the switch is engaged. An important point to remember is that although a distinct response is required to initiate the event, there is absolutely no way to determine (by the switch response alone) whether the screen event is understood or, for that matter, in what way the screen event is understood.

Try inserting the following routines as shown below. These routines are designed to illustrate a variety of BASIC statements when used in conjunction with a single switch program. Before adding each routine, delete lines 80 to 220.

```
DEL 80,220
```

Routine #1: VTAB & HTAB Screen Position

```
80 V = INT(RND(1)*24+1)
90 H = INT(RND(1)*40+1)
100 VTAB V: HTAB H
110 INVERSE
120 PRINT "*"
130 NORMAL
```

Routine #2: FLASH Screen Characters

```
80 V = INT(RND(1)*24+1)
90 H = INT(RND(1)*40+1)
100 VTAB V: HTAB H
110 FLASH
120 PRINT "!"
130 NORMAL
```

Routine #3: Upper-case ASCII Characters

```
80 C = C+1
90 IF C > 26 THEN C = 1
100 L$ = CHR$(64+C)
110 PRINT L$" ";
```

Routine #4: ASCII Numbers

```
80 C = C+1
90 IF C > 10 THEN C = 1
100 L$ = CHR$(47+C)
110 PRINT L$" ";
```

Routine #5: ON X GOTO Routines

```
80 C = C+1
90 IF C > 3 THEN C = 1
100 ON C GOTO 110,130,150
110 PRINT "1"
120 GOTO 230
130 PRINT "2"
140 GOTO 230
150 PRINT "3"
```

Routine #6: Screen Movement

```
80 HOME
90 L$ = "AUGMENTATIVE COMMUNICATION"
```

```
100 H = H+1
110 IF H > 40 THEN H = 1
120 VTAB 10
130 HTAB H
140 PRINT " "L$
```

Routine #7: Low-resolution Graphics

```
80 GR
90 R = INT(RND(1)*15+1)
100 COLOR = R
110 FOR K = 0 TO 39
120 HLIN 0,39 AT K
130 NEXT K
```

Routine #8: Low-Res VLIN and HLIN Commands

```
80 GR
90 P = INT(RND(1)*40)
100 C = INT(RND(1)*15+1)
110 COLOR=C
120 IF RND(1) > .5 THEN 150
130 VLIN 0,39 AT P
140 GOTO 230
150 HLIN 0,39 AT P
```

Routine #9: Displaying Low-res Color Screens

```
80 GR
90 FOR K = 0 TO 39
100 C = INT(RND(1)*15+1)
110 COLOR= C
120 HLIN 0,39 AT K
130 NEXT K
```

Routine #11: Low-resolution kaleidoscope:

```
10 GR
20 FOR L = 1 TO 3
30 FOR K = 0 TO 39
40 R = INT(RND(1)*15+1)
50 COLOR=R
60 HLIN 0,39 AT K
70 HLIN 0,39 AT 39-K
80 VLIN 0,39 AT K
90 VLIN 0,39 AT 39-K
100 NEXT K
110 NEXT L
```

Routine #10: High-resolution Graphics

```
35 HGR
80 C = INT(RND(1)*7+1)
90 HCOLOR=C
100 RR = INT(RND(1)*160)
110 RC = INT(RND(1)*280)
120 HPLOT RC,RR
```

Single Switch Input Techniques

There are many different techniques that can be used to input a single switch response. The SWITCH COLOR program shown below uses the low-resolution graphics capability of the Apple in conjunction with a typical single switch input format to change the screen color displayed each time the switch is engaged.

```
10 REM SWITCH COLOR
20 REM
30 HOME
40 GR
50 IF PEEK(-16287) > 127 THEN 50
60 IF PEEK(-16287) > 127 THEN 90
70 IF PEEK(-16384) = 155 THEN 150
80 GOTO 60
90 C = C+1: IF > 15 THEN C = 1
100 COLOR=C
110 FOR J = 1 TO 39
120 HLIN 1,39 AT J
130 NEXT J
140 GOTO 50
150 TEXT: HOME
160 POKE -16368,0
170 END
```

The GR statement in line 40 indicates that low-resolution graphics rather than normal screen text is used. The low-resolution color is determined in line 100 where **COLOR** is set to the value of C. For low-resolution graphics there are 16 possible color codes ranging from 0 (black) to 15 (white).

The SWITCH COLOR program presents the 15 different color codes as determined by line 90. Each time the switch is activated, the variable C is incremented by 1 and the screen color is set to C in line 100. The program continues to run until the Esc key is pressed. Chapter 3 provides many additional low-resolution single switch software applications.

Using the Keyboard as Switch

As was shown with the DEMO program, PEEK(-16384) can be used to determine whether or not a key has been pressed by scanning the keyboard for a possible response. This feature can also be used with the SWITCH COLOR program so that any key can be used to provide switch input.

For some individuals unable to use a keyboard in the traditional manner, the ability does exists to use the SPACEBAR or to simply hit one of the keyboard keys when a switch response is required. The following lines are used to scan the keyboard and if a key is pressed, the key code is stored in variable KY. If KY is greater than 127, control is sent to line 85 and the response is interpreted as a switch response.

```
70 KY = PEEK(-16384): IF KY > 127 THEN 85
85 POKE -16368,0
```

After a keyboard key has been pressed, the value in location PEEK(-16384) is not changed until a new key is pressed or until the statement in line 85 sets the keyboard strobe to a value 127 or less.

The variable KY could be used to either exit the program, or to sense a switch response as shown by these modifications:

```
70 KY = PEEK(-16384): IF KY = 155 THEN 150
75 IF KY > 127 THEN 85
85 POKE -16368,0
```

A specific keyboard key could be designated as the source of single switch input by modifying line 75. For example, the code for the SPACEBAR is 160. The following modification causes the SPACEBAR to be treated as a switch response (in addition an actual switch response).

```
75 IF KY = 160 THEN 85
```

To display the keyboard values corresponding to each key as the various keys are pressed, enter the following:

```
86 PRINT KY
```

Response Prompts

The above program can be modified to first display a program title, and then prompt the student to press the switch in order to begin the task:

```
31 VTAB 7: HTAB 15
32 PRINT "SWITCH COLOR"
33 VTAB 20: HTAB 9
34 PRINT "(PRESS SWITCH TO CONTINUE)"
35 IF PEEK(-16287) < 128 THEN 35
```

The switch response is read via a single line loop: the program leaves line 35 only when a response has been detected. In other words, as long as the value in PEEK(-16287) is less than 128, line 35 is read continuously. When the value in PEEK(-16287) is greater than 127, control is sent to the next program line.

A convention seems to have developed among single switch programmers to use the negative for referencing accessing memory locations to sense single switch input: PEEK-(16287) (the most frequently used) and PEEK(-16286). Instead of using the negative value, these same registers can be identified by using PEEK(49249) and PEEK(49250) so that:

```
PEEK(-16287) = PEEK(49249)
PEEK(-16286) = PEEK(49250)
```

If the statement in a program is using PEEK(49249), this is the same as using location PEEK(-16287) or the open-Apple key. The actual memory location for registers shown as a negative can be found by adding 65536 to the negative so that -16287 + 65536 = 49249.

```
DECIMAL LOCATION = 65536 + (-16287)
```

A joystick can also be used to provide single switch input. First connect the joystick to the nine-pin game port. With the COLOR program still in memory, add line 65:

```
65 IF PDL(0) < 50 THEN 90
```

The PDL(0) function reads the left-right movement of the joystick. If the joystick is in center position, the joystick sends a decimal value of about 125 to the computer. If the joystick is moved far left, a value of 0 if sent; and if the stick is moved far right, a value of 255 is sent. If necessary, adjust the axis trim control for the degree of joystick sensitivity required.

To read either a left or right movement, change line 65 as follows:

```
65 IF PDL(0) < 50 OR PDL(0) > 200 THEN 90
```

While PDL(0) reads the left-right joystick movement, PDL(1) reads the up-down movement. Moving the stick forward decreases the value of PDL(1), and moving the stick backward (which is a manageable movement for some students) increases this value. The following results in a switch response when the stick is moved in a backward direction:

```
65 IF PDL(1) > 200 THEN 90
```

Add line 66 to see the value of PDL(1) as the stick is being moved:

```
66 VTAB 22: PRINT PDL(1)"    "
```

Displaying the contents of PDL(0) and/or PDL(1) at the bottom of the screen while a programming is running can be use when setting the joystick trim controls. With the above modifications in place, note the value displayed by PDL(1). Now adjust the up-down trim control and see how this value varies. In most situations, the trim controls should be set so that the value displayed by PDL is approximately 127.

To include up-down as well as left-right joystick movements, the following could be added:

```
65 IF PDL(0) < 50 OR PDL(0) > 200 THEN 90
66 IF PDL(1) < 50 AND PDL(0) > 200 THEN 90
```

Notice also that one of the joystick buttons is the same as engaging the open-Apple key or PEEK(-16287).

If desired, a Touchwindow can be used to provide single switch input. The PDL function is used to read Touchwindow input. However, when a Touchwindow is connected to an Apple, the beginning value returned by PDL is 5. If the left side of the window is touched, the value returned is between 5 and 125. The value returned by the Touchwindow increases as the window senses input from left to right.

Line 65 is set to read Touchwindow input as follows:

```
65  IF PDL(5) > 5 THEN 90
```

The following table describes the various methods for sensing single switch input.

Switch Input Mode	**Switch Type**
PEEK(-16287) or PEEK(49249)	**switch #1, open-Apple key, switch button**
PEEK(-16286) or PEEK(49250)	**switch #2, closed-Apple key, switch button**
PEEK(-16384) or PEEK(49152)	**keyboard input, Esc key input,**
PDL(0)	**paddle #1, joystick, Touchwindow,**
PDL(1)	**paddle #2, joystick, Touchwindow**

Switch Access Problems

If a single switch device is connected to the nine-pin game port, and the software is compatible with the switch wiring, there should have no problem running the single switch software application. If the switch is wired to PEEK(-16287) and the software is using PEEK(-16286) to read switch responses, the switch or the software must be changed. Use the INPUT CHECK program described below to determine what software switch source is being used by the switch device connected to the game port, and then list the program to determine what switch is being used by the software application.

If an adaptive firmware card is being used, several things can happen to prevent a software application from running. If the switch is connected to the adaptive firmware card, and the software application runs continuously without so much as a look at the switch, be sure that nothing is connected to the nine-pin game port. Also, make sure that the Adaptive Firmware Card has been set for either the NORMAL or SW INPUT mode from the extended menu.

If an Apple II+ is being used, single switch input is not sensed as with more advanced Apple models. When a program is run with an Apple II+, the program might very well run continuously as if a switch is being continuously engaged. And this is exactly what is happening.

If PEEK(-16287) is being used with a IIe, IIc, or IIgs when switch #1 is engaged, a value greater than 127 is sent to the computer. The opposite is true for the II+. When a switch is engaged for this model, a value less than 128 is sent which means that the current value in PEEK(-16287) is already greater than 127.

If using a II+, potential input problems can be solved by changing the switch input statement as follows:

```
60 PEEK(-16287) > 127 THEN 90
```

should become

```
60 PEEK(-16287) <  128 THEN 90       (II+ single switch input)
```

The switch color program described above would be changed as shown below when using an Apple II+:

```
10 REM SWITCH COLOR II+
20 REM
30 HOME
40 GR
50 IF PEEK(-16287) < 128 THEN 50
60 IF PEEK(-16287) < 128 THEN 90
70 IF PEEK(-16384) = 155 THEN 150
80 GOTO 60
90 C = C+1: IF > 15 THEN C = 1
100 COLOR=C
110 FOR J = 1 TO 39
120 HLIN 1,39 AT J
130 NEXT J
140 GOTO 50
150 TEXT: HOME
160 POKE -16368,0
170 END
```

If a switch is not available and a II+ is being used, use the keyboard to provide switch input by adding these lines:

```
75 IF PEEK(-16384) > 127 THEN 85
85 POKE -16368,0
```

If an application is being used with both a II+ and a more recent Apple model, the following modification will indicate whether a II+ is being used; that is, if register 64435 is not equal to 6, then the Apple being used is a II+. This information can be used to set the switch for either a II+ or IIe/IIc/IIgs models:

```
42 IF PEEK(64435) < > 6 THEN 50
43 IF PEEK(-16287) > 127 THEN 43
```

```
44 IF PEEK(-16287) > 127 THEN 90
45 IF PEEK(-16384) = 155 THEN 150
46 GOTO 44
50 IF PEEK(-16287) < 128 THEN 50
60 IF PEEK(-16287) < 128 THEN 90
70 IF PEEK(-16384) = 155 THEN 150
80 GOTO 60
90 C = C+1: IF > 15 THEN C = 1
100 COLOR=C
110 FOR J = 1 TO 39
120 HLIN 1,39 AT J
130 NEXT J
140 GOTO 50
150 TEXT: HOME
160 POKE -16368,0
170 END
```

Continuous Activity Response

For the most part, the ability to initiate a screen activity by means of a discrete response is generally the preferred response. However, this does not mean that there is not a place for routines that allow for a continuous response. For beginning users, any type of response, be it discrete or continuous, should be allowed. The single switch user must first understand that there is a relation between engaging a switch and a screen activity. After this relationship has been demonstrated by observing the individual engage the switch to initiate a screen activity, the switch response can be restricted to require discrete responses.

To develop the ability to engage a switch for an extended period of time, a continuous activity or reverse switch response can be used. As shown in the modified SWITCH COLOR program below, line 115 has been added so that the screen activity stops when the switch is disengaged. This modification encourages the student to hold the switch in the closed position until the screen event has been completed.

```
10 REM SWITCH COLOR
20 REM
30 HOME
40 GR
90 C = C+1: IF > 15 THEN C = 1
100 COLOR=C
110 FOR J = 1 TO 39
115 IF PEEK(-16287) < 128 THEN 115
116 IF PEEK(-16384) = 155 THEN 150
120 HLIN 1,39 AT J
130 NEXT J
140 GOTO 90
150 TEXT: HOME
160 POKE -16368,0
170 END
```

The point at which the action is interrupted when the switch is disengaged can be further specified by enclosing the switch response in line 115 in a loop:

```
114 FOR D = 1 TO 5
116 NEXT D
```

The above continuous activity response (i.e., a response is required to continue the screen activity) is easily changed to a reverse activity response in which the switch is engaged to stop the activity by modifying line 115.

```
115 IF PEEK(-16287) > 127 THEN 115
```

The are really a great many types of switch responses that can be specified within a program. In single switch programming there is a place for every type of switch response. What must be done when using single switch software is to always consider the type of switch response that best meets the learning needs of the individual.

Controlled Program Interrupt

When working with a student, especially a student who is highly distracted, there might be an occasion when it is desirable to disable the keyboard and switch so that either a switch or keyboard response is ignored. The routine contained in lines 81 and 88 illustrate how this can be incorporated in a program. After the keyboard is scanned by means of line 70, program control is sent to line 81. If KY is equal to 155 (the Escape key value), the program run is terminated. However, if KY is equal to 140 (or the Control key + the L key), the screen is cleared and the program is set to hold until the Control+L key combination is pressed a second time.

The only key that results in the controlled program interrupt is the Control+L combination. This combination was selected because of the placement of these two keys on the keyboard which would not likely be pressed in combination (usually!) by a student banging away at keys at random. All other keyboard keys are treated as a switch-type response.

```
10 REM SWITCH COLOR
20 REM
30 HOME
40 GR
50 IF PEEK(-16287) > 127 THEN 50
60 IF PEEK(-16287) > 127 THEN 90
70 KY = PEEK(-16384): IF KY > 127 THEN 81
81 IF KY = 155 THEN 150
82 GR: HOME
83 FOR D = 1 TO 1000: NEXT D
84 POKE-16368,0
85 KY = PEEK(-16384)
86 IF KY = 140 THEN 88
87 GOTO 85
88 POKE-16368,0
90 C = C+1: IF > 15 THEN C = 1
100 COLOR=C
110 FOR J = 1 TO 39
120 HLIN 1,39 AT J
130 NEXT J
140 GOTO 50
150 TEXT: HOME
160 POKE -16368,0
170 END
```

BASIC Switch Summary

The following is a summary of the switch access methods discussed in this section. This list is by no means exhaustive and but it does provide a description of a variety of techniques used to read single switch input:

1. Program holds at line 100 if switch is closed.

```
100 IF PEEK(-16287) > 127 THEN 100
110 PRINT "CONTINUE"
```

2. Program holds at line 100 if the switch is open.

```
100 IF PEEK(-16287) < 128 THEN 100
110 PRINT "CONTINUE"
```

3. Program holds at line 100 if the switch is open.

```
100 S = PEEK(-16287)
110 IF S < 128 THEN 100
```

4. If switch is engaged, S is set to 1.

```
100 S = PEEK(-16287) > 127
110 PRINT PEEK(-16287)
120 IF S = 0 GOTO 100
```

5. Switch must first be open before a switch response can be made to send program control to line 130.

```
100 IF PEEK(-16287) > 127 THEN 100
110 IF PEEK(-16287) > 127 THEN 130
120 GOTO 110
130 PRINT "CONTINUE"
```

6. Switch must be opened before the screen event which follows a switch response is initiated.

```
100 IF PEEK(-16287) > 127 THEN 130
120 GOTO 100
130 IF PEEK(-16287) > 127 THEN 130
140  PRINT "CONTINUE"
```

7. A switch response sends control to line 120.

```
100 IF PEEK(-16287) > 127 THEN 120
110 GOTO 100
120 PRINT "CONTINUE"
```

8. Either the open- or closed-Apple switch sends control to line 120.

```
100 IF PEEK(-16287) > 127 THEN 120
105 IF PEEK(-16286) > 127 THEN 120
110 GOTO 100
120 PRINT "CONTINUE"
```

9. A variation of the above routine.

```
100 IF PEEK(-16287) > 127 OR PEEK(-16286) > 127 THEN 120
110 GOTO 100
120 PRINT "CONTINUE"
```

10. Either a switch or keyboard response sends control to line 120.

```
100 IF PEEK(-16287) > 127 OR PEEK(-16384) > 127 THEN 120
110 GOTO 100
120 PRINT "CONTINUE"
```

11. Either the open- or closed-Apple key or the joystick (moving the stick to the left) sends control to line 120.

```
100 IF PEEK(-16287) > 127 THEN 120
105 IF PEEK(-16286) > 127 THEN 120
106 IF PDL(0) < 75 THEN 120
110 GOTO 100
120 PRINT "CONTINUE"
```

12. A switch response sends control to line 120, while an Esc response

bypasses the single switch routine beginning in line 120.

```
100 IF PEEK(-16287) > 127 THEN 120
105 IF PEEK(-16384) = 155 THEN 130
110 GOTO 100
120 PRINT "CONTINUE"
130 POKE-16368,0
```

13. A variation of the above routine by using variable KY to store keyboard responses.

```
100 IF PEEK(-16287) > 127 THEN 120
105 KY = PEEK(-16384): IF KY = 155 THEN 130
110 GOTO 100
120 PRINT "CONTINUE"
130 POKE -16368,0
```

14. Variable SW is used to indicate input from register -16287.

```
100 SW = -16287
101 IF PEEK(SW) > 127 THEN 120
```

15. Alternative switch input (each switch must be engaged in alternating order to provide switch input).

```
100 IF SW = -16287 THEN SW = -16286: GOTO 105
101 SW = -16287
105 IF PEEK(-16384) = 155 THEN 130
106 IF PEEK(SW) > 127 THEN 120
107 GOTO 105
120 PRINT "SWITCH = "SW
125 GOTO 100
130 POKE-16368,0
```

16. Switch and keyboard responses are first stored in variable form before program control is determined.

```
100 S1 = PEEK(-16287)
101 S2 = PEEK(-16286)
102 S3 = PEEK(-16384)
110 IF S1 > 127 OR S2 > 127 OR S3 > 127 THEN 130
120 GOTO 100
130 PRINT "CONTINUE"
```

17. The switch must be engaged while in the loop otherwise the loop begins anew.

```
100 FOR D = 1 TO 100
110 IF PEEK(-16287) > 127 THEN 150
120 NEXT D
130 PRINT "NEW LOOP"
140 GOTO 100
150 PRINT "CONTINUE"
```

18. Variation of the above but using two switches.

```
100 FOR D = 1 TO 100
110 IF PEEK(-16287) > 127 THEN 150
115 IF PEEK(-16286) > 127 THEN 150
120 NEXT D
130 PRINT "NEW LOOP"
140 GOTO 100
150 PRINT "CONTINUE"
```

19. Variation of the above but an Esc response ends the program while in the loop.

```
100 FOR D = 1 TO 100
110 IF PEEK(-16287) > 127 THEN 150
115 KY = PEEK(-16384): IF KY = 155 THEN 160
```

```
120 NEXT D
130 PRINT "NEW LOOP"
140 GOTO 100
150 PRINT "CONTINUE"
160 END
```

20. A loop enclosed switch response but an actual counter is used instead of a FOR/NEXT loop.

```
100 IF PEEK(-16287) > 127 THEN 170
110 NC = NC+1
120 IF NC = 100 THEN 140
130 GOTO 100
140 PRINT "NEW LOOP"
150 NC = 0
160 GOTO 100
170 PRINT "CONTINUE"
```

21. The switch sensing statement is enclosed in a loop to provide information as to the duration of the switch response. The response interval in this returns a value between 0 and 250.

```
10 FOR D = 1 TO 250
20 IF PEEK(-16287) > 127 THEN RI = RI+1
30 NEXT D
40 PRINT "RESPONSE INTERVAL = "RI
```

22. The switch response is read as a subroutine (lines 1000 and 1010) in which program control remains in the subroutine until the switch is engaged.

```
100 GOSUB 1000
110 PRINT "CONTINUE"
120 END
1000 IF PEEK(-16287) < 128 THEN 1000
1010 RETURN
```

23. Variation of the above routine.

```
100 GOSUB 1000
110 PRINT "CONTINUE"
120 END
1000 IF PEEK(-16287) > 127 THEN 1020
1010 GOTO 1000
1020 RETURN
```

24. The program ends when variable SW has been set to 1 by engaging the switch while in the loop in the subroutine.

```
100 SW = 0
110 GOSUB 1000
120 IF SW = 1 THEN 140
130 END
140 PRINT "CONTINUE"
150 GOTO 100
1000 FOR D = 1 TO 100
1010 X = PEEK(-16287)
1020 IF X > 127 THEN 1050
1030 NEXT D
1040 GOTO 1060
1050 SW = 1
1060 RETURN
```

Error Detection Techniques

Most errors can be detected during a test run. Indeed, always test run a program before actually using the program in earnest. When the program is run, simple errors such as syntax should be corrected immediately following each error prompt. How many times should a program be run and tested? The answer is simple: until there are no more errors!

If an error occurs that is not easily detected, one of the following techniques might be useful. The following TEST program listed below is used to illustrate what might be considered more advanced debugging techniques.

Variable Checks

An old standard in debugging is to stop the execution of a program at various points to inspect program variables. When the TEST program is run, switch input is used to count from 1 to N (or when the Esc key is pressed). Run and test the program to make sure that it is working.

```
10 REM TEST
20 REM
30 HOME
40 X = X+1
50 VTAB 10: HTAB 18
60 PRINT X
70 IF PEEK(-16287) > 127 THEN 70
80 IF PEEK(-16384) = 155 THEN 120
90 SW = PEEK(-16287)
100 IF SW > 127 THEN 30
110 GOTO 80
120 POKE -16368,0
130 END
```

After the initial test run, enter the following statement:

```
95 STOP
```

Now when the program is run, the number 1 appears center screen but the program stops with the following message:

```
BREAK IN 95
```

List the program and then display the contents of the X variable and the SW variable by:

```
?X,SW
1              32
```

The values displayed on screen indicate that 1 is in variable X and 32 is in variable SW. Knowing the contents of a variable is often useful when attempting to discern why a program is not doing what it should be doing. For example, if a Y had been inadvertently entered in line 60 (e.g., PRINT Y), the value displayed on screen would be a 0, but the value of X would reveal a value of 1. In this situation, checking X would indicate that either X was not incremented or displayed as expected.

If STOP's are used within a program when debugging, be sure to delete these statements when the program is actually used. On more than one occasion I have used a single switch program with a student only to have the program interrupted because of a STOP inserted during a debugging process.

ONERR GOTO Traps

Another programming technique that can be used to detect errors is to incorporate ONERR traps. The ONERR GOTO XXX statement indicates that when an error occurs, the program goes to the line specified. To illustrate how this works, delete the STOP in line 95 (if this has not already been done), and enter the following:

```
10 REM TEST-2
20 REM
25 ONERR GOTO 112
112 VTAB 12
113 PRINT "ERROR = "PEEK(222)
114 PRINT "LINE # = "PEEK(218) + PEEK(219)*256
115 IF PEEK(-16287) > 127 THEN 115
116 POKE 216,0
117 CLEAR
118 IF PEEK(-16287) > 127 THEN 25
119 GOTO 118
```

Line 25 stipulates that if an error occurs, program execution is transferred to line 112. The number of the error that occurs is contained in register 222. The error number is displayed and the line number where the error occurred (lines 113 and 114). After the switch is released, the ONERR function is discontinued and all variables cleared before the program begins anew in line 25.

To see how the error trap works, add this line:

```
50 VTAB 10+X*3: HTAB 18
```

Activate the switch several times and the following should appear:

```
ERROR = 53
LINE # = 50
```

Examples of ONERR statement usage can also be found in the CATALOG/UTILITY and DISK MENU programs described in the next section. Although the ONERR is extremely useful to trap errors, deactivating this function when debugging or test running a program is sometimes necessary in order to detect errors. In other words, a syntax error is better than having this type of error suppressed and the program branching to a specific program location each time an error occurs.

The POKE 216,0 in line 116 above resets the program so that normal error messages are generated. If this were not done, an error anywhere in the program would always send control back to the line designated after the ONERR GOTO statement.

The following is a list of ONERR code values and corresponding errors:

Code	Error
0	NEXT without FOR
16	Syntax error
22	RETURN without GOSUB
42	Out of DATA error
53	Illegal quantity error
77	Out of memory error
107	Bad subscript error
163	Variable type mismatch

TRACE

The TRACE command can be used to help locate the source of an error by printing the line number of the program as each line is executed.

Clear memory with NEW and then enter the following three statements. Assume that this is a portion of a program and line 250 should direct control to line 300 and not back to line 250.

```
250 IF PEEK(-16287) > 127 THEN 250
260 GOTO 250
300 END
```

When this program is run, nothing happens because the program is "hung-up" at line 250. This problem is very visible here, but in an extremely long program this type of error might not be so easily detected.

Of course, for the above type of problem, the difficulty involves a logical error rather than a BASIC programming error. In other words, the program is doing exactly what it was programmed to do.

To exit the program, use Control+C. Next, enter TRACE and press RETURN and then re-run the program:

```
TRACE
RUN
```

When the program is run with the TRACE in operation, the screen is filled with the program line numbers:

```
#250 #250 #250 #250 #250 #250 #250 #250
```

As shown, only line number #250 appears because when the switch is engaged, control is continuously directed back to line 250.

Use Control+C to interrupt the program, change line 250 so that control is sent to line 300 and then run the program again. Now when the switch is engaged line #300 should be the last line number to appear in the TRACE.

```
#250 #260 #250 #260 #250 #260...#250 #300
```

To turn off TRACE, enter NOTRACE and press RETURN:

```
NOTRACE
```

The TRACE command can be very useful when looking for hard to detect bugs, but if this feature is used while accessing a disk file DOS becomes deactivated. This problem, for the most part, can be solved by adding CHR$(13) which, as mentioned previously, is the CHR$ equivalent to the RETURN key to the DOS D$ variable. Enter the following to see how TRACE works in conjunction with a DOS command:

```
NEW
10 D$ = CHR$(13) + CHR$(4)
20 PRINT D$;"CATALOG"
TRACE
RUN
```

Error Proofing

In addition to the ONERR command described above, certain programming precautions can be taken to eliminate possible errors. For example, assume that an INPUT statement is used to specify scan speed:

```
10 INPUT "SCAN SPEED (1=FAST TO 9=SLOW): ";SP
```

If a value less than 1 or greater than 9 is entered, the value of SP can be set to designated minimum or maximum limits:

```
20 IF SP < 1 THEN SP = 1
30 IF SP > 10 THEN SP = 9
```

When the above line is executed and a letter key pressed, the following prompt appears:

```
?REENTER
```

This occurs because an alphabetic character cannot be entered as an arithmetic value. The possibility of this error occurring can be avoided by entering a string value and then changing the string value to an arithmetic value as follows:

```
11 INPUT "SCAN SPEED (1=FAST TO 9 =SLOW): ";SP$
12 SP = VAL(SP$)
```

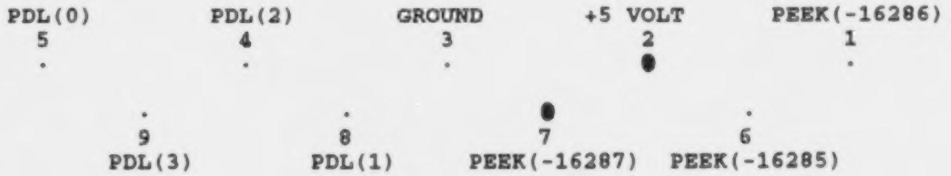
Now, if an alphabetic character is entered, the string variable is converted to an arithmetic variable. If the string is not a number value, SP is set to 0.

Utility Programs

Input Check

Knowing the whereabouts of the different switch locations provides an opportunity to verify that switch input is being received by the computer. Run the below program and the screen displays exactly what is being received when a switch or joystick is engaged, or when a keyboard key is pressed. If a switch is connected to the game port, the display shows a value greater than 127 then the switch is engaged. If nothing happens, check the switch and 9-pin connection. If a switch is not available, press the open Apple key and the SWITCH #1 line displays a value of 160.

Most single switch devices are connected via the nine-pin game port located on the back of the Apple (or the 16 pin input device located on the inside of the II+ and IIGS). The following shows the paddle and switch locations of the various pins. To connect a switch device using PEEK(-16287), one wire is connected to the +5 volt pin (#2) and a second wire to pin number #7. The following illustrates the pin locations as viewed from the back of a nine-pin male plug. When the single switch device is engaged, contact is made between pins #2 and #7 and the value in register PEEK(-16287) is set to a value of 128 or greater.



The above nine-pin specifications can also be used to build a switch. For example, by soldering the ends of two different 24 gauge 2 conductor audio cable wires to pins 2 and 7, and then soldering the other ends of these wires to the corresponding switch terminals, a switch is easily constructed so that when engaged, a value greater than 127 is registered in memory location -16287 (see Appendix C for IBM PC pin specifications):.

Switch devices such as mercury switches are relatively inexpensive and easily built. In addition, a variety of plate and tread switches can also be constructed. In the case of tread switches, I have built several using a switch called a Treadlite (#T-51-S) and sold by the Linemaster Switch Corporation located Woodstock, Connecticut and distributed by dealers selling electrical supplies. However, with this type of switch the spring must be replaced with one requiring less tension.

Switch input is also available by means of the 16 pin internal game connector. On the 16 pin connector, pin #1 is the +5 volt source and pin #2 corresponds to PEEK(-16287) or the open-Apple key.

The INPUT CHECK program can be used to check Touchwindow, joystick input, and to determine whether a switch is connected via location PEEK(-16287) or PEEK(-16286). Finally, use the program to make sure that each keyboard key is working properly. Each time a key is pressed, the corresponding keyboard character (either lower- or uppercase) is shown at the top of the screen. The keyboard code or ASCII value corresponding to each keyboard character is also shown.

When the Esc key is pressed, the display simply shows that this key is functioning. To really "escape" from this program, use the CONTROL+C combination.

```
10 REM INPUT CHECK
20 REM
30 HOME
40 FOR K = 1 TO 40: L$ = L$ + "-": NEXT K
50 VTAB 3: HTAB 15
60 PRINT "INPUT CHECK"
70 E$ = "     "
80 VTAB 13: PRINT L$
90 VTAB 10: HTAB 21
100 PRINT "VALUE IS < 255 WHEN"
110 HTAB 21
120 PRINT "JOYSTICK IS ENGAGED."
130 VTAB 8: PRINT L$
140 VTAB 18: PRINT L$
150 VTAB 15: HTAB 21
160 PRINT "VALUE IS > 127 WHEN"
170 HTAB 21
180 PRINT "SWITCH IS ENGAGED."
190 POKE -16368,0
200 KY = PEEK(-16384)
210 C$ = CHR$(KY)+E$
220 IF KY = 27 THEN C$ = "ESC"
230 IF KY < 27 THEN C$ = "C+" + CHR$(64 + KY)
240 IF KY = 8 THEN C$ = "<--"
250 IF KY = 13 THEN C$ = "RET"
260 IF KY = 21 THEN C$ = "-->"
270 VTAB 6: PRINT "KEY = "C$
280 VTAB 6: HTAB 12
290 PRINT "CODE = "KY"  "
300 P1 = PDL(0)
310 S1 = PEEK(-16287)
```

```
320 S2 = PEEK(-16286)
330 P2 = PDL(1)
340 VTAB 10
350 PRINT "PADDLE #1 = "P1;E$
360 PRINT "PADDLE #2 = "P2;E$
370 VTAB 15
380 PRINT "SWITCH #1 = "S1;E$
390 PRINT "SWITCH #2 = "S2;E$
400 GOTO 190
410 END
```

Disk Access

When attempting to interrupt a program in order to list the program or to display the catalog and the program re-boots each time Control+Reset is pressed, the HELLO program probably contains the following statement which makes Control+Reset the command to boot the program.

```
POKE 1012,0
```

In order to find out whether this is the case or not, exit the disk menu if possible and load the HELLO program. If there is no way to exit the disk menu, boot the system with a different disk, and then load the HELLO program from the disk which re-boots when Control+Reset is pressed. List the program and determine whether a line contains POKE 1012,0. Delete this line and then re-save the HELLO program. Now, when Control+Reset is pressed, disk programs can be interrupted.

Of course if a simple method is wanted to prevent others from interrupting a program using Control+Reset, insert POKE 1012,0 at the beginning of the HELLO program.

```
5 POKE 1012,0
```

If POKE 1012,0 is used and there is a need to reset Control+Reset so that this sequence operates in the usual manner, the following changes Control-Reset back to its normal function:

```
POKE 1012,56
```

Catalog Utility Programs

Instead of immediately booting a disk and running a program, a catalog utility can be installed as the greeting program so that the disk CATALOG is displayed each time the program is booted. At the end of the catalog a routine can be added that inputs a program name and then run the program specified.

To install the HELLO UTILITY program enter the program as shown, or load HELLO-U from the program disk (use NEW to clear memory before loading this program). After the program has been loaded, save the program using the greeting program name:

```
LOAD HELLO-U
SAVE HELLO-U

10 REM HELLO UTILITY
20 REM
30 L$ = "SINGLE SWITCH BASIC PROGRAM"
40 L$ = LEFT(L$,38)
```

```
50 HOME: VTAB 2
60 PRINT L$
70 ONERR GOTO 90
80 GOTO 160
90 HOME: POKE 216,0
100 VTAB 7
110 PRINT "ERROR = "PEEK(222)
120 VTAB 15: HTAB 8:
130 PRINT "(PRESS RETURN TO CONTINUE)";
140 POKE -16368,0: GET KY$
150 GOTO 50
160 PRINT CHR$(4);"CATALOG"
170 HTAB 8: PRINT "EXIT"
180 VP = PEEK(37)
190 IF VP < 20 THEN 220
200 FOR L = 1 TO 4: PRINT: NEXT L
210 VP = 19
220 VTAB VP+4: HTAB 3
230 PRINT "(ENTER PROGRAM NAME AND PRESS RETURN)"
240 VTAB VP+1
250 PRINT: PRINT SPC(6)">";
260 POKE -16368,0
270 INPUT "";PN$
280 IF PN$ < > "EXIT" THEN 300
290 POKE -16368,0: VTAB 24: END
300 PRINT
310 PRINT CHR$(4);"RUN"PN$
```

An example of how to use the above utility program might be to include a series of readiness programs on a single disk. Line 30 can be changed to something like READINESS ASSESSMENT PROGRAMS, a new disk initialized with the above utility as the greeting program, and then programs saved that deal with single switch readiness skills and various program modifications.

Greeting Program Switch Control

There are several techniques that can be used to read switch control values when the Apple is first booted and then to use these control values when a single switch application is run. As an example, the scan speed could be read by means of the greeting program and all subsequent programs (or specified programs) would use these values.

The following modifications to the HELLO UTILITY program (or to whatever greeting program is being used on a disk) inputs a scan speed value and name, store both on disk in a file called SWITCH CONTROL. Other programs on the disk can retrieve the information in the SWITCH CONTROL file, and then use these values within the application.

```
21 HOME: VTAB 5
22 INPUT "NAME: ";N$
23 PRINT
24 SCAN SPEED (1 TO 9): ";SP
25 D$ = CHR$(4)
26 PRINT D$;"OPEN SWITCH CONTROL"
27 PRINT D$;"WRITE SWITCH CONTROL"
28 PRINT N$: PRINT SP
29 PRINT D$;"CLOSE SWITCH CONTROL"
```

The information in the SWITCH CONTROL file is retrieved by the following routine which can be added to the beginning lines of the single switch application:

```
21 D$ = CHR$(4)
22 PRINT D$;"OPEN SWITCH CONTROL"
23 PRINT D$;"READ SWITCH CONTROL"
24 INPUT N$: INPUT SP
25 PRINT D$;"CLOSE SWITCH CONTROL"
```

A similar technique is described in Chapter 8 in the section on Creating/Modifying Database Files in which a reading program imports a scan speed value from a program entitled DOS CONTROL.

If a single value is specified in the greeting program, a quick-and-easy technique can be used to store the value in an unused memory location, and then retrieve this value after a single switch application has been run. For example, the scan speed could be specified in the greeting program, stored in memory location 950, and then retrieved when an application on the disk is run.

```
21 HOME: VTAB 5
22 INPUT "SCAN SPEED (1 TO 9): "SP
23 POKE 950,SP
```

The scan speed value stored in memory location 950 is easily recalled within a program by

```
21 SP = PEEK(950)
```

Disk Menu

If the DISK MENU program listed below is used, up to 14 disk programs can be displayed when the program is run. To use this program, first enter the program and then modify the DATA statement in line 410 to include whatever programs that are listed when the disk is first booted. If a program name not on the disk is specified, an ONERR statement is used to send control to line 350 where the error number is printed. If no error is encountered, the program specified is run from the disk.

The title of the disk can be reset by changing variable N$ in line 40. In line 160, the program name is automatically centered and then displayed. The following is an example of the type of screen format that appears when the disk is booted:

```
                 SINGLE SWITCH PROGRAMS
----------------------------------------------
             1 = SWITCH
             2 = READ
             3 = SPELLTALK
SELECTION:       (E=EXIT TO BASIC)
----------------------------------------------
     (Make Selection and Press RETURN)
```

To run one of the programs shown in the menu, all that needs to be done is to enter the number that corresponds to the program name and then press RETURN.. The program automatically assigns a number to each of the program names listed in the DATA statement in line 410.

```
10 REM DISK MENU
20 REM
30 DIM NP$(14)
40 N$ = "SINGLE SWITCH PROGRAMS"
```

```
50 FOR K = 1 TO 40: H$ = H$ + "-": NEXT K
60 ONERR GOTO 110
70 FOR K = 1 TO 14
80 READ NP$(K)
90 VP = VP+1
100 NEXT K
110 POKE 216,0
120 HOME
130 VTAB 3: PRINT H$
140 VTAB 21: PRINT H$
150 INVERSE: VTAB 2
160 HTAB 21-LEN(N$)/2
170 PRINT N$
180 VTAB 5
190 NORMAL
200 FOR K = 1 TO VP
210 H = 13: IF K > 9 THEN H = 12
220 HTAB H: PRINT K " = "NP$(K)
230 NEXT K
240 VTAB 24: HTAB 5: PRINT "(Make a Selection and Press
    RETURN)";
250 VTAB VP+6: HTAB 1: PRINT "SELECTION: ";
260 VTAB VP+6: HTAB 17: PRINT "(E=EXIT TO BASIC)"
270 VTAB VP+6: HTAB 12
280 IF VP < 10 THEN PRINT " ";
290 POKE -16368,0: INPUT "";C$
300 NC = VAL(C$)
310 IF C$ = "E" OR C$ = "e" THEN END
320 IF NC < 1 OR NC > VP THEN 120
330 ONERR GOTO 350
340 GOTO 400
350 HOME: VTAB 10: PRINT "DISK ERROR: "PEEK(222)
360 POKE 216,0
370 VTAB 15: HTAB 8: PRINT "(PRESS RETURN TO CONTINUE)";
380 POKE -16368,0: GET R$
390 GOTO 120
400 PRINT CHR$(4);"RUN "NP$(NC)
410 DATA SWITCH,COLOR,READ,MATH,SPELLTALK
```

The DISK MENU program can be used by first loading the program into memory, inserting a blank disk into drive #1, and then initializing the disk using the DISK/MENU program as the HELLO or greeting program. When this is done, and when the disk just initialized is booted, the menu appears.

The DISK MENU can be used to create a variety of "turnkey" applications (or systems that simply require the user to "turn the key," or in this case, to "turn the computer on" to get started). Thus, a separate disk might be used for readiness programs, another for scan applications, and another for academic tasks.

HELLO/DISK MENU Applications

The DISK MENU program can be saved as the HELLO program so that each time the disk is booted, the disk menu appears. For an already initialized disk, enter the DISK MENU Program and then save as the HELLO program:

SAVE HELLO (where DISK MENU is the program in memory)

or initialize a blank disk with DISK MENU in memory:

INIT HELLO

To move from an application program back to the DISK MENU, the use the following in place of the END statement or whatever procedure is used to exit the application:

```
XXX PRINT
XXX PRINT CHR$(4);"RUN DISK MENU"
```

If the greeting program is also used to load binary file applications such as a speech synthesizer (see Chapter 4) or a character font set (see Chapter 6), the HELLO program might load and/or run the binary files, and then run the DISK MENU program. Now when the application program is exited and the DISK MENU program is re-run, the binary program is not re-loaded into memory. In most situations, loading or running an Applesoft program will not destroy a binary program in memory.

```
10 REM HELLO
20 REM
30 HOME
40 D$ = CHR$(4)
50 PRINT D$;"BRUN TEXTALKER"
60 PRINT D$;"BLOAD LASCII"
70 PRINT D$;"RUN DISK MENU"
```

System Master Utilities

The DOS 3.3 System Master disk contains several useful utility programs. In addition to the COPYA program already discussed, this disk contains a program called RENUMBER. Load this program into memory, the save the program on the program disk using the name RENUM:

```
LOAD RENUMBER

SAVE RENUM
```

With the RENUM program saved on the program disk, you might want to begin computer sessions which will involve BASIC programming by first running this utility. After RENUM has been run and the RENUM instruction screen displayed, press RETURN. The message RENUMBER IS INSTALLED AND READY should appear.

```
RUN RENUM
```

When RENUM is run, the line numbers within a program can be renumbered so that all the lines are incremented by 10 by entering & and pressing RETURN. This utility is useful for cleaning up a program after many modifications. The size of the line number increments between program lines can also be modified by using the RENUM utility. To set the first line number of a program to 10, and to increment the program lines by 20, the following is used:

```
&FIRST 10, INC 20
```

By just entering the symbol **&** and pressing RETURN, the first line of a program is automatically set to 10 and all lines are incremented by 10.

The RENUM program is very useful for making some sense out program line numbers that have been modified and inserted beyond recognition. This utility is also very useful for creating space for additional line numbers. After modifying a program by adding lines, and making sure that the program works, enter **&** and you will have ample space for additional insertions.

For those wishing to combine programs, RENUM is an essential utility.

To combine two programs, enter the first program or routine and then enter **&H** to hold the program in memory while a second BASIC program is loaded and run. Modify the second program or routine so that the program lines are in concert with the program on hold. You do not want to duplicate program lines. Next, enter **&M** to merge the two programs. With a little practice the RENUM merge and hold functions can be a real time saver.

One final RENUM note. If you use certain programs such as the TEXTALKER for Echo applications, this will likely bomb the RENUM utility. Should this happen, re-run RENUM or even re-start the system if necessary.

Another useful utility on the System Master disk is called FID and is run by entering BRUN rather than just RUN:

```
BRUN FID
```

This program contains a menu of programs for copying files, determining how much space is on a disk, and several other useful options

APPLE Memory

For those interested in the how the Apple's memory is related to BASIC, the following section might be of some use. If not, skip this section and your ability to use the applications described in the subsequent chapters will not be hampered.

The Apple's memory is divided into Random Access Memory (RAM), Read-Only Memory (ROM), and Input/Output Locations. For the single switch BASIC user, the RAM portion of the Apple's memory is what is really important.

The following are BASIC instructions provide information concerning how the Apple's memory is configured and the amount of programming memory that is currently available. To determine the amount of free programming space available use the FRE function:

```
PRINT FRE(0)
-32020
```

The above is especially useful for determining the amount of memory available for very large BASIC programs. In the above example, the value returned by FRE(0) is -32020. If the value returned is negative, use the following to print the positive decimal equivalent of available memory:

```
PRINT 65536 + FRE(0)
33516
```

To determine the first memory location of an Applesoft program use the following:

```
PRINT PEEK(103) + PEEK(104) * 256
2049
```

But why is it necessary to multiply one of the PEEKS's by 256? The 65,536 memory locations in the 6502 chip (which is the heart of the Apple) are numbered from 0 to 65535. Each of these locations can store a single decimal value, and this must be a number from 0 to 255. The value 255 (which seems to appear frequently in BASIC programming) is equivalent to the binary value 11111111 or hexadecimal value of FF. Because each register or memory location contains only one byte, and because one byte is equal to 8 bits (e.g., the binary value 11111111 is comprised of 8 bits), the maximum value that can be stored in a memory location is 255 or

128X1 + 64x1 + 32x1 + 16x1 + 8x1 + 4x1 + 2x1 + 1x1 = 255

To keep track of a memory location or address having a value greater than 255, a second byte of memory is used. In determining the starting location of an Applesoft program, PEEK(103) indicates the values 0 to 255, and PEEK(104) the value in excess of 255 (or 256 to 65280). Thus, the highest memory location is 255 + 255 X 256 = 65535. When Applesoft is first booted, PEEK(103) is set to 1 and PEEK(104) to 8 which signifies a starting memory location of 1 + 8 X 256 = 2049. To determine the last memory location of an Applesoft program use the following:

```
PRINT PEEK(175) + PEEK(176) * 256
2769
```

As can be seen, when the system is first booted, the beginning programming location is 2049. If high-resolution graphics are not used, the program and variables used can extend up to location 38400 which is called the high memory location or HIMEM. The value of HIMEM indicates the highest memory location available to an Applesoft program. The actual HIMEM value depends on the amount of memory available and is found by

```
PRINT PEEK(115) + PEEK(116) * 256
38400
```

String arrays begin at HIMEM and extend downward. If a program contains a large number of string statements (e.g., a series of sentences stored in an array containing 50 strings (e.g., S$(1), S$(2)...S$(50)), the contents of these string array variables would be stored at a point beginning with HIMEM, and extend downward in terms of program entry.

If high-resolution graphics are used, and page 1 (i.e., the primary page picture buffer) of high-resolution begins at location 8192, the program can overlap into space dedicated to high-resolution graphics and problems can occur. In other words, if a BASIC program extends into space used by high-resolution graphics, the graphics can bomb the BASIC instructions in this memory area. When page 1 high-resolution graphics are in use, the usable size of an Applesoft BASIC program is:

8192 - 2049 = 6143

which is approximately 24 sectors of disk space. If a program is partially destroyed when the HGR command is used, one remedy might be to use HGR2 or page 2 (i.e., the secondary page picture buffer) which sets the high resolution graphics to full screen (280 columns by 192 rows). Although HGR2 increases the amount of available programming space, there are several additional techniques that can be used to accommodate large programs and thus avoid complications which might arise from using secondary page graphics.

Simple variables used by a BASIC program begin at a point called LOMEM which, unless otherwise set, immediately follow the BASIC program in memory. Depending upon the types of variables used, programming space can be saved by setting LOMEM to either 16384 (if only page 1 high-resolution graphics are used) or 24576 (if both page 1 and page 2 high-resolution graphics are used):

```
LOMEM: 16384
or
LOMEM: 24576
```

The current value of LOMEM is found by the following:

```
LOMEM: 16384
PRINT PEEK(105) + PEEK(106) * 256
16384
```

A second method for accommodating very long BASIC programs, and one which provides a considerable amount of extra memory if page 1 high-resolution graphics are being used, is to change the starting point of the BASIC program in memory. The following program resets the beginning Applesoft program starting location from the usual starting address of 2049 when the system is first booted to the address immediately after the last location of page 1 of high-resolution graphics or 16384:

```
10 A = 16384
20 POKE A - 1,0
30 POKE 103, A - INT(A/256) * 256
40 POKE 104, INT (A/256)
```

When the above program is run (be sure to save it on disk first), Applesoft is reset to begin at location 16384 so that a fairly large program can be run without interfering with page 1 of high-resolution graphics. If both page 1 and page 2 high-resolution graphics are being used, set the starting address to 24576.

```
10 A = 24576
```

If the beginning program address is set to 16384, the size of a program and all variables used can be

38400 - 16384 = 22016

which is a substantial amount of memory that can be dedicated to a program when hi-res graphics are in use. A variation of this programming technique is used in the MEMORY MOVE described in the Chapter 3.

Hexadecimal Numbers

Every once in a while you will come across a memory location that is a combination of letters and numbers. These are called hexadecimal numbers or just "hex" numbers and will look something like the following: $D000, $C061, $FF, $FFFF. The hexadecimal numbering system consists of 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Instead of 10, 11, 12, 13, 14, and 15, hex values are expressed as A, B, C, D, E and F. When used to indicate computer memory, hex values are designated as such by the dollar sign ($) symbol.

Decimal	Hex
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E

15	**F**

For combinations of hex values, the base 16 number system is used to translate the values to decimal equivalents. Thus, $FFFF is equal to (15 X 4096) + (15 X 256) + (15 X 16) + 15 or 65535; the value $C061 is equal to 49249 which is the equivalent of -16287. Although you may never use a hex value, there might be an occasion when it is useful knowing what these values mean and even what memory location is being referenced.

Statement Finder

The following section deals with a technical aspect of BASIC and can be avoided with no harm done to either your knowledge for or appreciation of single switch software design, use, or development. In other words, unless you have a compelling interest in the really basic aspects of BASIC, skip this section. However, if you do have an interest in advanced single switch BASIC applications, the Statement Finder routine described below might be of some interest.

As was said before, the Apple 6502 chip or "microprocessor" has a total of 65,536 memory locations. Each memory location is identified by a number called an address, and can hold one byte which is used to represent characters or instructions. Clear memory and enter and run this one line program:

```
10 PRINT 4+6
10
```

Address	Value	Meaning
2051	10	Line number
2052	0	
2053	86	PRINT
2054	52	4
2055	200	+
2056	54	6

When this one line program is run, the contents of each memory location is executed and the sum of 4 + 6 is printed. Notice that the code for PRINT is the numeric value 186. When PRINT is used in a program, the instruction is stored as the decimal 186 in one of the Apple's address registers (address 2053 in the above program).

The all-important PEEK(-16287) is stored in Apple registers as the decimal values 226,40,49,54,50,56,55,41 (where the decimal value 226 is the decimal equivalent of PEEK).

The decimal equivalents of BASIC statements stored in resisters can be useful when working with fairly long programs and there is a need to identify all the PEEK's within the program or some other BASIC statement. The following is used by adding the seven line routine to the beginning of a BASIC program. Line 1 contains the decimal equivalent of the PDL instruction. When the program is run with the STATEMENT FINDER routine added, every line number containing a PDL instruction is displayed.

```
1 X = 226
2 S = PEEK(103) + PEEK(104) * 256
3 E = PEEK(175) + PEEK(176) * 256
4 FOR J = S+144 TO E: IF PEEK(J) > 0 THEN 7
5 L = PEEK(J+3) + PEEK(J+4) * 256
6 J = J+4
7 IF PEEK(J) = X THEN PRINT L" ";
```

```
8 NEXT J: END
```

The above routine can be used to find whatever statement is specified in line 1. For example, the decimal equivalent of the PDL command is 216. The STATEMENT FINDER routine was added to a fairly long single switch program containing over 500 lines with many lines having compacted instructions.

Instead of attempting to find every PDL by listing the program (and it's easy to overlook commands when reviewing listings), the Statement Finder routine was added to the program by first running RENUM, then the Statement Finder Program, and then using &H to hold the routine in memory. The single switch application was then loaded and the two programs merged. When run, the program quickly identified the line numbers containing PDL instructions.

```
1640  2130  2240  2460  3660
```

The purpose of identifying the PDL instruction was to change the type of joystick movement sensed from PDL(0) to PDL(1). With the help of the Statement Finder routine, this is accomplished without a great deal of difficulty.

Another application might be a program that has a number of STOP statements buried in the code as a result of various debugging efforts. Setting X in line 1 to 179 and then running the program will quickly reveal all the STOP statements contained in the program. REM statements could be identified by setting X to 178, POKE's by setting X to 185, and PEEK's by setting X to 226.

The Statement Finder routine is primarily deigned for use with longer programs, but it can be a useful aide for identifying statements that need to be modified (e.g., changing PEEK's from -16286 to -16287) and to identify program bugs (e.g., identifying all the loops within a program beginning with the FOR statement). The Applesoft BASIC programming reference manual contains a complete list of statement codes (p. 121).

Chapter 3

Low-resolution Graphics

The Apple computer has two primary graphics modes: low-resolution and high-resolution graphics. Low-resolution graphics can display up to 1,024 screen blocks using a screen comprised of up to 40 columns and 48 rows. High-resolution can display up to 53,760 dots using a screen comprised of up to 280 columns and 192 rows.

The advantage of low-resolution graphics in comparison to high-resolution graphics is that it uses much less memory and disk space to store a low-resolution graphics image; the advantage of high-resolution graphics is in the name: the graphics image can be much clearer because of the greater "resolution" or the number of dots used to make the image.

The following is a list and several examples of the primary of low-resolution BASIC instructions:

GR	Set to low-resolution graphics mode.
COLOR= 15	Set the color to white.
PLOT 0,0	Display a dot in column 0 and row 0.
PLOT 1,20	Display a dot in column 1 and row 20.
PLOT 5,16	Display a dot in column 5 and row 16.
HLIN 15,25 AT 20	Draw a horizontal line from column 15 to 20 at row 20.
VLIN 5,10 AT 30	Draw a vertical line from row 5 to row 10 in column 30.
TEXT	Return to text or regular non-graphics screen mode.

The LOWRES SAMPLE program illustrates the use of the COLOR and PLOT instructions. When this program is run, a simple shape appears on the screen. Each time the switch is engaged, the color of the low-resolution is changed. By adding, deleting or modifying shape statements an infinite number of low-resolution screen images can be created.

```
10 REM LOWRES SAMPLE
20 REM
30 GR
40 IF PEEK(-16287) > 127 THEN 40
50 IF PEEK(-16287) > 127 THEN 80
60 IF PEEK(-16384) = 155 THEN 190
70 GOTO 50
80 C = C+1: IF C > 15 THEN C = 1
90 COLOR=C
100 PLOT 20,20
110 PLOT 19,21
120 PLOT 21,21
130 PLOT 18,22
140 PLOT 22,22
150 PLOT 17,23
160 PLOT 23.23
170 HLIN 0,39 AT 25
180 GOTO 40
190 TEXT
200 POKE -16368,0
210 END
```

The program begins by setting the graphics mode in line 30, senses switch input in lines 40-70, and then displays the low-resolution image in lines 80-170. The color is set in line 80 using variable C, beginning with color code #1, incrementing this value each time the switch is engaged, and then resetting this value to 1 if the color value in C exceeds 15. The color of the road is easily changed by adding line 165:

```
165 COLOR=15
```

The following is a list of the various low-resolution colors available. Although low-resolution graphics does not have the clarity of high-resolution graphics, there are twice as many colors. Experiment with the various low-resolution colors by changing line 90 or by adding additional COLOR statements.

Code	**Color**	**Code**	**Color**
0	**black**	8	**brown**
1	**magenta**	9	**orange**
2	**dark blue**	10	**grey 2**
3	**purple**	11	**pink**
4	**dark green**	12	**green**
5	**grey**	13	**yellow**
6	**medium blue**	14	**aqua**
7	**light blue**	15	**white**

When the program is run notice that the word RUN remains at the bottom of the screen display. This is because the first 20 rows of the screen are used for graphics and the last four for text. The entire screen can be set to low-resolution graphics by adding the following:

```
35 POKE -16302,0        (entire screen low-resolution graphics)

36 CALL -1998           (clears low-resolution screen to black)
```

Or the screen can be cleared at the beginning of the program by means of a HOME statement:

```
25 HOME
```

To exit the program, the Esc key is used. Press the Esc key and then enter LIST to list the program.

The LOWRES SAMPLE program can be modified by using the HLIN instruction to create an image of a car. To modify the program, replace lines 100 to 160 with the following:

```
100 HLIN 19,22 AT 18
110 HLIN 19,22 AT 19
120 HLIN 15,25 AT 20
130 HLIN 15,25 AT 21
140 HLIN 15,25 AT 22
150 HLIN 15,25 AT 23
160 PLOT 17,24: PLOT 22,24
```

Each time this program is run, a low-resolution image of a car appears when the switch is engaged. For each subsequent switch input, the car changes color (as does the road on which the car is placed). When the Esc key is pressed and the program run ended, the screen changes to inverted @ symbols indicating that the screen has been changed from low-resolution graphics to text mode. When the program is listed, the program code is displayed using the full screen.

Sequential Graphics

By changing the lines used to check for a switch response to a subroutine, as shown in the LOWRES-SG listing below, the program can be modified to display low-resolution components in a sequential graphics cause/effect format. When this is done, each switch response results in the display of a different low-resolution component. In this program, the switch subroutine is accessed by the GOSUB instruction. When the switch is engaged, the newly created subroutine (lines 40 to 80) senses the input, control is returned (as indicated by the RETURN in line 80) to the statement immediately following the GOSUB statement.

```
10 REM LOWRES-SG
20 REM
25 HOME
30 GR
35 GOTO 90
40 IF PEEK(-16287) > 127 THEN 40
50 IF PEEK(-16287) > 127 THEN 80
60 IF PEEK(-16384) = 155 THEN 180
70 GOTO 50
80 RETURN
90 GOSUB 40: COLOR=3
100 HLIN 19,22 AT 18
110 HLIN 19,22 AT 19
115 GOSUB 40: COLOR=7
120 HLIN 15,25 AT 20
130 HLIN 15,25 AT 21
140 HLIN 15,25 AT 22
150 HLIN 15,25 AT 23
155 GOSUB 40: COLOR=13
160 PLOT 17,24: PLOT 22,24
165 GOSUB 40: COLOR=15
170 HLIN 0,39 AT 25
171 GOSUB 40: COLOR=8
172 VLIN 7,24 AT 32
175 GOSUB 40: COLOR=12
176 PLOT 28,4: PLOT 30,2
177 PLOT 32,5: PLOT 34,7
178 PLOT 34,3
180 POKE -16368,0
190 END
```

Each time the subroutine is called (i.e., a switch response is entered), a component of the image is displayed in the color specified. The "tree" is displayed via the subroutine beginning in line 175.

After running the program with the above change, list the program. As can be seen, the program is listed but only the last four lines of the screen are used. The reason this happens is that a TEXT statement was not inserted so that the low-resolution image would remain on the screen after the program ended. In order to list the entire program, enter TEXT and press RETURN:

```
TEXT
```

The LOWRES-SG program displays each low-resolution screen component and then the program ends. The program routines can be placed in a continuous loop by adding line 179:

```
179 GOSUB 40: GR: GOTO 90
```

With this modification in place the program continually displays the low-resolution components in sequence until the Esc key is pressed.

Single Switch Readiness

Assessing Readiness

When working with young children or persons unfamiliar with computers and single switch devices, cause-effect programs provide an excellent opportunity to develop important readiness skills for more sophisticated learning and academic tasks. However, the term "cause-effect" can be somewhat misleading. For cause-effect (e.g., LOWRES SAMPLE) programs already described, an assumption is made that the individual realizes that a switch response "causes" a screen activity (e.g., the screen changing color, appearance of a face, sound, etc.). However, there is no way of knowing how the individual actually conceptualizes the relationship between the switch response and the screen activity.

When using cause-effect software, it is important encourage distinct and purposeful responses by using ample verbal cues and verbal reinforcement. In addition, it is also important to increase the complexity of the switch task when the student has developed sufficient readiness skills as indicated by the ability to exhibit distinct and purposeful responses.

The readiness scan program shown below illustrates one method for bridging the gap between cause-effect and scan software. This program is designed to encourage purposeful behavior by requiring the child to wait until the "nose" prompt appears. A cursor moves from left to right until the cursor is center screen and a "nose" prompt appears. If the switch is engaged before the prompt appears, no feedback is given. If engaged while the prompt is displayed, a face appears.

When using this program, verbal prompts such as "wait, or "don't press the switch yet" as the cursor moves from left to right should be used often. When the nose appears, indicate that a switch response should be made by "Now, press the switch." When the task is completed without the aid of verbal prompts, a reasonable assumption can be made that the individual has conceptualized the existence of a relationship between engaging the switch and the screen activity (i.e., the appearance of Nilt's face).

```
10 REM NILT'S NOSE
20 REM
30 HOME
40 GR
50 COLOR = 3: PLOT 4,15
60 FOR K = 4 TO 19 STEP 3
70 COLOR=3
80 PLOT K,15
90 FOR L = 1 TO 100
100 IF PEEK(-16287) > 127 THEN 380
110 IF PEEK(-16384) = 155 THEN 400
120 NEXT L
130 COLOR=0: PLOT K,15
140 NEXT K: COLOR=3
150 PLOT 20,15
160 PLOT 19,16: PLOT 21,16
170 FOR D = 1 TO 100
180 IF PEEK(-16287) > 127 THEN 210
190 NEXT D
200 GOTO 380
210 HLIN 15,25 AT 5
```

```
220 PLOT 14,6: PLOT 13,7
230 PLOT 26,6: PLOT 27,7
240 VLIN 8,24 AT 12
250 VLIN 8,24 AT 28
260 PLOT 13,25: PLOT 14,26
270 PLOT 15,27: PLOT 16,28
280 PLOT 27,25: PLOT 26,26
290 PLOT 25,27: PLOT 24,28
300 HLIN 17,23 AT 29
310 COLOR=12
320 PLOT 17,10: PLOT 18,10
330 PLOT 17,11: PLOT 18,11
340 PLOT 22,10: PLOT 23,10
350 PLOT 22,11: PLOT 23,11
360 COLOR=9
370 HLIN 17,23 AT 22
380 FOR L = 1 TO 1500: NEXT L
390 GOTO 40
400 TEXT: HOME
410 POKE -16368,0
420 END
```

By adding line 381, NILT's face remains on the screen as long as the switch is engaged:

```
381 IF PEEK(-16287) > 127 THEN 381
```

The above program certainly lacks pizazz, but this can be remedied by adding a sound subroutine. First, enter the machine language program to generate the sound. This is accomplished adding a sound routine in lines 2, 4 and 6. A detailed discussion of sound and speech enhancements is provided in Chapter 4.

```
2 FOR K = 1 TO 21: READ A
4 POKE 801+K,A: NEXT K
6 DATA 174,32,3,173,48,192,136,208,5,206,
  33,3,240,6,202,208,245,76,34,3,96
```

Now add lines 45, 46 and 47 to give the scan sound as it moves from left to right:

```
45 POKE 800,200-K*4
46 POKE 801,100
47 CALL 802
```

Next, add the GOSUB to play the musical tune after a "correct" response:

```
355 GOSUB 5000
```

Last, add the routine that plays the different notes:

```
5000 POKE 800,101
5010 POKE 801,48
5020 CALL 802
5030 POKE 800,76
5040 POKE 801,48
5050 CALL 802
5060 POKE 800,60
5070 POKE 801,48
5080 CALL 802
5090 POKE 800,52
5100 POKE 801,96
```

```
5110 CALL 802
5120 POKE 800,60
5130 POKE 801.48
5140 CALL 802
5150 POKE 800,52
5160 POKE 801,255
5170 CALL 802
5180 RETURN
```

Refer to Chapter 4 for a detailed discussion concerning how to modify the POKE's in the above subroutine to vary the sound/music production.

Measuring Latency

Latency is an important variable to consider when evaluating an individual's ability to use a single switch device and when determining the appropriate scan duration for each individual. To illustrate how latency can be measured, the loop used to evaluate the switch can also be used to provide latency data. In line 170, a loop comprised of 100 iterations is used to evaluate a switch response. In other words, D is first set to 1 and PEEK(-16287) is evaluated, then D is set to 2 and PEEK(-16287) is evaluated again. This is repeated until D has been set to 100. When the program task concludes, the value of D provides information as to how soon the switch was engaged following the appearance of Nilt's nose (i.e., the target stimulus).

In the following program additions, the variable N is used to keep track of the number of number of items and D is the value in the loop that was reached just prior to engaging the switch:

```
385 N = N+1
386 PRINT N" "D
387 FOR L = 1 TO 1500: NEXT L
388 HOME
```

Because variable D provides an index of the time available to respond to the target prompt, the value of D can be reported as a percentage. Thus, if D is 79, and the maximum size of the loop is 100, D can be printed as a percentage of 100 or:

```
386 PRINT N" LATENCY = "INT(D/100*100+.5)
```

The .5 in the above statement is used to round off to the next highest percentage.

If the loop size has been set previously, the statement in 386 can be modified as follows:

```
35 SP = 150
170 FOR D = 1 TO SP
386 PRINT N" LATENCY = "INT(D/SP*100+.5)
```

The scan speed is set by variable SP in line 35; SP signifies the number of scan iterations in line 170; and latency is then reported as a percentage in line 386. Because the scan duration is fairly long in the above modification, it might be useful to reduce the number of cursor scans that appear prior to the appearance of the stimulus target.

Scan Tracking

The ability to generalize is critical in most educational tasks. If a student is able to respond successfully using NILT'S NOSE, the following SCAN TRACKING program can be used to provide a conceptually similar program but one in which the format is slightly different. For this program, a cursor moves from left to right. If the switch is activated when the cursor is within the boundaries of the box displayed on the screen, sound and visual feedback is given.

The number of trials presented for each run is determined by the value of N set in line 40. The program is terminated after N items have been presented or by pressing Esc.

```
10 REM SCAN TRACKING
20 REM
30 SP = 5
40 N = 10
50 FOR K = 1 TO 21
60 READ A
70 POKE 801+K,A
80 NEXT K
90 DATA 174,32,3,173,48,192,136,208,5,206,
   33,3,240,6,202,208,245,76,34,3,96
100 GOTO 300
110 POKE 800,101
120 POKE 801,48
130 CALL 802
140 POKE 800,76
150 POKE 801,48
160 CALL 802
170 POKE 800,60
180 POKE 801,48
190 CALL 802
200 POKE 800,52
210 POKE 801,96
220 CALL 802
230 POKE 800,60
240 POKE 801,48
250 CALL 802
260 POKE 800,52
270 POKE 801,255
280 CALL 802
290 RETURN
300 HOME
310 FOR X = 1 TO N
320 GR
330 FOR L = 1 TO 1200: NEXT L
340 COLOR=9
350 HLIN 12,27 AT 13
360 HLIN 12,27 AT 27
370 HLIN 13,27 AT 12
380 VLIN 13,27 AT 27
390 FOR K = 1 TO 12
400 POKE 800,150-K*6
410 POKE 801,150
420 CALL 802
430 COLOR=3
440 PLOT K*3+1,20
450 PLOT K*3+2,20
460 PLOT K*3+1,21
470 PLOT K*3+2,21
480 FOR L = 1 TO SP*10
490 IF PEEK(-16287) > 127 THEN 520
500 IF PEEK(-16384) = 155 THEN 700
```

```
510 NEXT L: GOTO 540
520 IF K > 3 AND K < 9 THEN 610
530 GOTO 680
540 COLOR=0
550 PLOT K*3+1,20
560 PLOT K*3+2,20
570 PLOT K*3+1,21
580 PLOT K*3+2,21
590 NEXT K
600 GOTO 690
610 GOSUB 110
620 FOR K = 1 TO 5
630 FOR L = 1 TO 750: NEXT L
640 COLOR=10+K
650 FOR J = 1 TO 14
660 VLIN 14,26 AT 12 + J
670 NEXT J: NEXT K
680 FOR L = 1 TO 1000: NEXT L
690 NEXT X
700 POKE -16368,0
710 TEXT: HOME
720 END
```

In the above SCAN TRACKING program a switch check is not used to prevent continuous responses. If the switch is pressed prior to the cursor entering the box, the screen is cleared and the next item presented. In other words, positive feedback is provided only if the switch is engaged when the cursor is in the designated area.

As with other scan tracking tasks of this type, verbal and/or physical prompts should be provided as needed to demonstrate what occurs when the cursor is in the box and the switch engaged. However, the ultimate goal is for a student to successfully engage the switch with no prompts whatsoever.

Low-resolution Applications

Game Activities

For students capable of a more challenging task, the concept underlying NILT'S NOSE can be used to create a things-that-fall-from-the-sky arcade game. For this program, seven blocks are displayed at the top of the screen. The blocks can be described as spaceships, aliens, etc. in the traditional arcade scenario. One of the blocks is randomly selected, and slowly moves from top to bottom. If the switch is engaged while the block is in the target area, the Apple beeps and a hit is scored.

```
10 REM ARCADE
20 REM
30 DIM V(40)
40 HOME: GR
50 COLOR=15
60 FOR K = 4 TO 36 STEP 5
70 FOR J = 1 TO 3
80 HLIN K,K+2 AT J
90 NEXT J: NEXT K
100 FOR K = 1 TO 7
110 R = INT(RND(1)*7+1)
120 IF V(R) = 1 THEN 110
130 V(R) = 1
140 R = R*4+(R-1)
```

```
150 FOR L = 1 TO 750: NEXT L
160 COLOR=2
170 VLIN 25,27 AT R-1
180 VLIN 25,27 AT R+3
190 COLOR=0: J = 1
200 FOR J = 0 TO 12
210 COLOR=15: GOSUB 390
220 FOR D = 1 TO 75
230 IF PEEK(-16287) > 127 THEN 250
240 NEXT D: GOTO 270
250 IF J = 8 THEN 290
260 GOTO 320
270 COLOR=0: GOTO 390
280 NEXT J: GOTO 320
290 FOR L = 1 TO 7
300 PRINT CHR$(7): NEXT L
310 COLOR=3: GOSUB 390
320 FOR L = 1 TO 2000: NEXT L
330 COLOR=0
340 VLIN 25,27 AT R-1
350 VLIN 25,27 AT R+3
360 NEXT K
370 TEXT: HOME
380 END
390 FOR L = 1 TO 3
400 HLIN R,R+2 AT J*3+L
410 NEXT L
420 RETURN
```

Joystick Graphics

If a student is able to use a joystick, most programs are easily modified to read joystick input. The left-right joystick movement is controlled by the BASIC statement PDL(0), and the up-down joystick is controlled by PDL(1).

When the joystick is first connected to the nine-pin game port, the computer reads the position of the horizontal and vertical axis by means of PDL(0) and PDL(1). If the joystick is in center position, PDL(0) and PDL(1) should return a value of approximately 125.

The horizontal and vertical axis input can be a value ranging from 0 to 255. If unsure about how a joystick reads left-right and up-down input, connect a joystick to the nine-pin game port and run the INPUT CHECK program described in Chapter 2. This program is also useful for determining whether the joystick is operating correctly and to adjust the axis trim controls which determine the initial value returned by the joystick when in the neutral position.

The JOYSTICK program first reads the PDL(0) and PDL(1) statements to determine if the joystick has been moved. If the joystick has been moved left, and P1 is less than 50, the value of H is reduced by 1. If the joystick is moved right, and P1 is greater than 200, 1 is added to H. The same principle is used to determine the vertical position of the joystick. If V and H are within acceptable limits (each must be a value between 0 and 39), a low-resolution dot is displayed at H,V. The program is terminated by pressing the Esc key.

```
10 REM JOYSTICK
20 REM
30 H = 20: V = 20
40 GR
```

```
50 COLOR=15
60 IF PEEK(-16384) = 155 THEN 190
70 P1 = PDL(0)
80 IF P1 < 50 THEN H = H-1
90 IF P1 > 200 THEN H = H+1
100 IF H < 0 THEN H = 0
110 IF H > 39 THEN H = 39
120 P2 = PDL(1)
130 IF P2 < 50 THEN V = V-1
140 IF P2 > 200 THEN V = V+1
150 IF V < 0 THEN V = 0
160 IF V > 47 THEN V = 47
170 PLOT H,V
180 GOTO 60
190 POKE -16368,0
200 TEXT: HOME
210 END
```

If the joystick displays dots at a speed faster than what a student can manage, a time delay loop can be inserted in line 175:

```
175 FOR D = 1 TO 200: NEXT D
```

To use the entire screen for low-resolution graphics, add lines 45 (this sets the screen to full graphics mode) and line 46 (this clears the entire screen to black):

```
45 POKE -16302,0
46 CALL -1998
```

Joystick Feedback

The following program provides feedback corresponding to the intensity of joystick movement. As the program is now set, the intensity of a "backward" or "pull" joystick response is sensed. The value of PDL(1) is displayed at the bottom of the screen. The degree to which the joystick is pulled provides corresponding sound and visual (i.e., low-resolution color) feedback. This program can be used to reinforce the relationship between joystick movement and a corresponding screen activity, and to assess the ability to use a joystick to initiate a screen activity.

```
10 REM JOYSTICK FEEDBACK
20 REM
30 HOME
40 FOR K = 1 TO 21
50 READ A
60 POKE 801+K,A
70 NEXT K
80 DATA 174,32,3,173,48,192,136,208,5,206,33,3,
   240,6,202,208,245,76,34,3,96
90 PS = PDL(1)
100 PI = (255-PS)/39
110 GR
120 XP = PDL(1)
130 VTAB 21: HTAB 18: PRINT XP"  "
140 IF PEEK(-16384) = 155 THEN 270
150 H = INT((XP-PS)/PI)
160 IF H = HX THEN 220
170 FOR L = 0 TO H
180 COLOR=INT(L/2.75)+1
190 IF L = 0 THEN COLOR=0
```

```
200 HLIN 0,39 AT 39-L: NEXT L
210 IF H < 1 THEN H = 1: GOTO 240
220 POKE 800,225-H*4
230 POKE 801,40: CALL 802
240 FOR L = 0 TO 39-H
250 COLOR=0: HLIN 0,39 AT L: NEXT L
260 HX = H: GOTO 120
270 TEXT: HOME
280 POKE -16368,0: END
```

Touchwindow Input

The program can be further modified to read Touchwindow input. Because the Touchwindow returns PDL(0) and PDL(1) values in the 0 to 255 range, these values must be reduced so that they are within the 0 to 39 low-resolution graphics range (lines 100-130).

```
10 REM TOUCHWINDOW SKETCH
20 REM
30 GR
40 POKE -16302,0
50 CALL -1998
60 COLOR=15
70 IF PEEK(-16384) = 155 THEN 160
80 P1 = PDL(0)
90 P2 = PDL(1)
100 P1 = INT(P1/5.14-5.85)
110 IF P1 < 1 OR P1 > 35 THEN 70
120 P2 = INT(P2/5.14-5.85)
130 IF P2 < 1 OR P2 > 35 THEN 70
140 PLOT P1,P2
150 GOTO 70
160 POKE -16368,0
170 TEXT: HOME
180 END
```

Many single switch programs can be converted for use with a Touchwindow by using the PDL(0) instruction. A Touchwindow might be especially useful when attempting to show that a direct response can cause a screen effect. A Touchwindow provides a very close connection between the response and screen, and thus can be used to focus attention to the screen task.

The Nilt's Nose program is easily modified for use by changing line 100 from:

```
100 IF PEEK(-16287) > 127 THEN 380
```

to

```
100 IF PDL(0) > 25 THEN 380
```

The above instruction serves as a switch so that touching the screen is interpreted as a single switch response. If desired, the exact screen location can be specified by using PDL(0) and PDL(1) to further delineate screen coordinates as shown by the following example:

```
100 IF PDL(0) > 75 AND PDL(0) < 200 PDL(1) > 75 AND PDL(1) < 200 THEN 380
```

When specifying a specific Touchwindow screen location, be sure that the designated screen space is of a size that the child or student can touch. Also, be sure that the Touchwindow location matches the actual computer screen

BEST COPY AVAILABLE

location. Additional techniques for using the Touchwindow are discussed in Chapters 4 and 5.

A Low-resolution Authoring System

An authoring system provides an easy method for creating a task or activity. The following is a very simple authoring system, but it does illustrate how a series of low-resolution images can be created for use in a single switch format.

```
10 REM LOWRES AUTHOR
20 REM
30 MX = 3
40 F$ = "D"
50 GR: HOME: N = 0
60 PRINT "(PRESS SWITCH TO BEGIN OR E FOR EDITOR)"
70 IF PEEK(-16287) > 127 THEN 70
80 IF PEEK(-16287) > 127 THEN 120
90 KY = PEEK(-16384): IF KY = 155 THEN 630
100 IF KY = 197 THEN 170
110 GOTO 80
120 N = N+1: IF N > MX THEN N = 1
130 PRINT
140 PRINT CHR$(4);"BLOAD"F$ + STR$(N)
150 GOTO 70
160 GR
170 CC = 15: HOME
180 PRINT "U=UP        D=DOWN   L=LEFT   R=RIGHT"
190 PRINT "E=ERASE     S=SAVE   G=GET    C=COLOR"
200 PRINT "Q=QUIT      W=WRITE  X=DON'T WRITE"
210 POKE -16368,0
220 H = 20: V = 20
230 COLOR = CC: PLOT H,V
240 X = PEEK(-16384)
250 IF X > 127 THEN 300
260 FOR L = 1 TO 8: NEXT L
270 IF CC = 0 THEN COLOR=15
280 IF CC > 0 THEN COLOR=0
290 PLOT H,V: GOTO 230
300 X$ = CHR$(X-128): POKE -16368,0
310 IF X$ = "Q" THEN 30
320 IF X$ = "E" THEN 160
330 IF X$ = "C" THEN 490
340 IF X$ = "S" OR X$ = "G" THEN 540
350 IF X$ = "W" THEN D = 0: GOTO 230
360 IF X$ = "X" THEN D = 1: GOTO 230
370 IF D = 1 THEN COLOR = 0: PLOT H,V
380 IF X$ = "U" THEN V = V-1
390 IF X$ = "D" THEN V = V+1
400 IF X$ = "L" THEN H = H-1
410 IF X$ = "R" THEN H = H+1
420 IF H < 0 THEN H = 0
430 IF H > 39 THEN H = 39
440 IF V < 0 THEN V = 0
450 IF V > 39 THEN V = 39
460 IF D = 1 THEN 230
470 PLOT H,V
480 GOTO 230
490 HOME: VTAB 23: HTAB 5
500 PRINT "(MAKE SELECTION AND PRESS RETURN)"
510 VTAB 21: INPUT "COLOR ( 1 TO 15)? ";CC$
```

```
520 CC = VAL(CC$)
530 GOTO 230
540 HOME
550 INPUT "FILE NUMBER? ";N$
560 FX$ = F$+N$
570 HOME
580 IF X$ = "G" THEN 610
590 PRINT CHR$(4);"BSAVE";FX$",A1024,L1024"
600 GOTO 620
610 PRINT CHR$(4);"BLOAD";FX$
620 FX$ = "": GOTO 170
630 POKE -16368,0
640 TEXT: HOME
650 END
```

When the LOWRES AUTHOR is run, first create a series of low-resolution images so that each image, or part of an image, is stored on disk in a separate file. When the program is run, each switch response loads and displays each file in the file sequence.

Before creating files, set variable MX to the number of low-resolution images specified in line 30. As the program is now written, the program displays the contents of three low-resolution files with each file name beginning with a **D** (see line 40). To use a file name other than "D," set F$ to the desired name in line 40.

In order to edit a low-resolution screen, press E after the prompt. The various editing commands are shown at the bottom of the screen. Remember that if MX has been set to 3 in line 30, three files are created by entering 1, 2 and then 3 after the FILE NUMBER PROMPT?

The low-resolution screen is saved on disk in line 590 where **A** indicates the beginning address of the file and **L** signifies the length of the file.

Screen Inversion

The low-resolution screen can be inverted by changing screen locations containing a low-resolution dot to black, and all locations that are black to a color. This is done to invert low-resolution screens or to create low-resolution graphics by entering black low-resolution dots on an all-white screen (use the X or don't write command). The low-resolution screen inversion consists of the following modifications:

```
305 IF X$ = "I" THEN 1000
1000 FOR K = 1024 TO 2039
1010 MC = PEEK(K)
1020 IF MC = 0 THEN MC = CC*17: GOTO 1060
1030 IF MC < 16 THEN MC = CC*16: GOTO 1060
1040 IF INT(MC/16) = MC/16 THEN MC = CC: GOTO 1060
1050 MC = 0
1060 POKE K,MC: NEXT K
1070 GOTO 170
```

Low-resolution Screen File Names

Low-resolution screens are saved as binary files on disk. Each file requires approximately 6 sectors of space and has a catalog entry similar to the following:

```
B 006 YES
```

The low-resolution graphics image YES in the file YES (there is a difference) was created using the LOWRES AUTHOR program.

Instead of using the LOWRES AUTHOR to save files in a sequence of files, specific file names can be created in order to be used with other programs. By changing line 550 and deleting line 560, each file is saved using the exact file name specified:

```
550 INPUT "FILE NAME? ";FX$
560
```

If the program disk is being used, make the above modification and then run the LOWRES AUTHOR program. Enter E to edit a file, and then enter YES following the FILE NAME prompt.

Although using low-resolution files is a rather slow task, the following easy-to-write routine illustrates how a low-resolution file such as **YES** file is loaded and displayed each time the switch is engaged.

```
10 REM LOWRES SCREEN
20 REM
30 FX$ = "YES"
40 HOME
50 GR
60 COLOR=2
70 IF PEEK(-16287) > 127 THEN 70
80 IF PEEK(-16287) > 127 THEN 110
90 KY = PEEK(-16384): IF KY > 127 THEN 110
100 GOTO 80
110 POKE -16368,0
120 IF KY = 155 THEN 160
130 PRINT CHR$(4);"BLOAD ";FX$
140 FOR L = 1 TO 3500: NEXT L
150 GOTO 50
160 TEXT: HOME
170 END
```

When the above program is run, the low-resolution file specified in line 30 is loaded and displayed each time the switch is engaged. The low-resolution screen appears for approximately five seconds (as determined by the timing loop in line 140) before the screen is cleared. Press Esc to exit the program.

The color of the screen can be changed by modifying line 60. The following change results in a different color each time the low-resolution image is displayed.

```
60 C = C + 1: IF C > 15 THEN C = 1
65 COLOR=C
```

Low-resolution Memory

The above program is a bit cumbersome because the low-resolution screen file is loaded and displayed each time the program is run. A more efficient method is to load the file once, and then switch back and forth between the low-resolution image and a second, albeit blank, low-resolution screen.

This can be accomplished, but not without some difficulty. The Apple computer actually has two low-resolution screens referred to as page 1 and page 2. The problem with using page 2 low-resolution graphics is that it is difficult to access because this is where BASIC programs are initially stored.

In order to use both low-resolution pages, the initial memory starting point for storing BASIC programs must be changed. This can be accomplished by entering the following program and then saving the program on disk (don't run the program just yet):

```
10 REM MEMORY MOVE
20 REM
30 HOME
40 VTAB 2: PRINT "MEMORY MOVE ROUTINE:"
50 VTAB 7
60 PRINT "OLD STARTING ADDRESS:";
70 PRINT PEEK(103) + PEEK(104) * 256
80 PRINT: PRINT
90 INPUT "NEW STARTING ADDRESS: ";A
100 POKE A-1,0
110 POKE 103, A-INT(A/256)*256
120 POKE 104, INT(A/256)
130 END
```

Run the above program and then enter 24576 as the starting address for BASIC programs following the prompt NEW STARTING ADDRESS. Now run the modified LOWRES SCREEN-2 program:

```
10 REM LOWRES SCREEN-2
20 REM
30 HOME
40 FX$ = "YES"
50 GR
60 PRINT CHR$(4);"BLOAD";FX$
70 POKE -16299,0
80 POKE -16302,0
90 COLOR=2
100 IF PEEK(-16287) > 127 THEN 100
110 IF PEEK(-16287) > 127 THEN 140
120 KY = PEEK(-16384): IF KY > 127 THEN 140
130 GOTO 110
140 POKE -16368,0
150 IF KY = 155 THEN 200
170 POKE -16301,0
160 POKE -16300,0
180 FOR L = 1 TO 3500: NEXT L
190 GOTO 70
200 TEXT: HOME
210 END
```

To return to the BASIC starting address that is used when the system is first booted (i.e., 2049), enter **FP** and press RETURN:

For most low-resolution single switch applications, there is no need to change the starting memory address for BASIC programs. However, for high-resolution applications which can require a considerable amount of memory, changing the starting address is required in order to prevent memory overlap between the BASIC program and the memory used by the high-resolution graphics screen. When this occurs, a good chunk of the BASIC program currently in memory will "bomb." Consult Chapter 6 for a discussion concerning the use of the MEMORY MOVE program in order to increase BASIC programming memory when used with high-resolution graphics.

Soft Switches

The POKE's -16299,0 and -16302,0 in the LOWRES SCREEN-2 program are seen frequently in single switch programs and are referred to as "soft switches" in that each time the values are poked the software makes the appropriate graphics switch. In LOWRES SCREEN-2 program, POKE -16299,0 causes the program to switch from page 1 to page 2 graphics; likewise, POKE-16300,0 causes the program to switch from page 2 to page 1 graphics.

The following is a list of various soft switches, the decimal equivalent, and a very brief description concerning how each is used. Using soft switches within a program does take a bit of practice, especially regarding various switch combinations, but the following should give some idea as to the purpose of a soft switch when encountered in a single switch program.

POKE -16297,0 or POKE 49239,0	(switch to high-resolution graphics)
POKE -16298,0 or POKE 49238,0	(switch to low-resolution graphics)
POKE -16299,0 or POKE 49237,0	(switch to page 2 graphics)
POKE -16300,0 or POKE 49236,0	(switch to page 1 graphics)
POKE -16301,0 or POKE 49235,0	(switch to split-screen graphics)
POKE -16302,0 or POKE 49234,0	(switch to full screen graphics)
POKE -16303,0 or POKE 49233,0	(switch from graphics to text)
POKE -16304,0 or POKE 49232,0	(switch from text to graphics)

Chapter 4

Sound and Speech

Built-In Apple Sound

Easy-to-Use Sound Functions

Whether it is a beep, a musical tune, or speech, sound can be an excellent enhancement for many programs. Likewise, being able to disable the sound component might be just as important. The following section describes how sound can be used to enhance single switch programs (or any other program for that matter), and how to modify the sound routine to produce whatever sound is desired. Needless to say, one's ability to orchestrate sound and music is much dependent on one's knowledge of sound and music.

The FACE program is used to illustrate how a very simple cause/effect single switch routine can be modified to incorporate sound and even music. Each time the switch is activated, part of a face is displayed on screen. The graphics can be changed to display a house, car, birthday cake, or whatever else might be of interest to the child.

```
10 REM FACE
20 REM
30 HOME
40 GR
50 COLOR=9
60 GOSUB 520
70 PLOT 20,15
80 PLOT 19,16: PLOT 21,16
90 GOSUB 520
100 COLOR=1
110 HLIN 15,25 AT 5
120 GOSUB 520
130 COLOR=13
140 PLOT 14,6: PLOT 13,7
150 PLOT 26,6: PLOT 27,7
160 GOSUB 520
170 VLIN 8,24 AT 12
180 VLIN 8,24 AT 28
190 GOSUB 520
200 PLOT 13,25: PLOT 14,26
210 PLOT 15,27: PLOT 16,28
220 PLOT 27,25: PLOT 26,26
230 PLOT 25,27: PLOT 24,28
240 GOSUB 520
250 COLOR=7
260 HLIN 17,23 AT 29
270 GOSUB 520
280 COLOR=12
290 PLOT 17,10: PLOT 18,10
300 PLOT 17,11: PLOT 18,11
310 GOSUB 520
320 COLOR=14
330 PLOT 22,10: PLOT 23,10
340 PLOT 22,11: PLOT 23,11
350 GOSUB 520
360 COLOR=9
```

```
370 HLIN 17,23 AT 22
380 FOR K = 1 TO 4
390 GOSUB 520
400 COLOR=0
410 PLOT 17,22: PLOT 23,22
420 COLOR=9
430 PLOT 17,21: PLOT 23,21
440 FOR L = 1 TO 750: NEXT L
450 COLOR=0
460 PLOT 17,21: PLOT 23,21
470 COLOR=9
480 PLOT 17,22: PLOT 23,22
490 NEXT K
500 TEXT: HOME
510 END
520 IF PEEK(-16287) > 127 THEN 520
530 IF PEEK(-16287) > 127 THEN 550
540 GOTO 530
550 RETURN
```

When the above program is run, each switch response displays a part of a face. After the complete face is displayed, each switch response causes the face to "smile." The FACE program listing shows how the low-resolution face is sequentially displayed following each switch response. The program is divided into 10 components so that each component begins with a switch input subroutine (GOSUB 520), followed by a low-resolution image segment. The 10 components begin with the following line numbers:

Line #	Function
60	nose
90	top line
120	top corners
160	vertical lines
190	bottom corners
240	bottom line
270	left eye
310	right eye
350	mouth
390	mouth movement

The Apple II+, IIe and IIc models do not have a great deal of easy-to-use sound capabilities. The primary attribute of these models is the ability to generate a beep or a click which is useful but not the stuff of which best sellers are made.

The beep or bell sound (think of the bell sound on an old typewriter) is produced by CHR$(7). Enter the following, press RETURN, and the Apple beep should be heard:

```
PRINT CHR$(7)
```

The beep can be added to the switch subroutine (lines 520 to 550) to enhance each switch response:

```
550 PRINT CHR$(7)
560 RETURN
```

The beep can also be enclosed in a loop so that each time the switch is activated to create a low-resolution smile, a series of beeps is sounded:

```
440 FOR L = 1 TO 5
```

```
445 PRINT CHR$(7): NEXT L
```

As was already said, the Apple is capable of producing a beep and a click. The click is sounded by PEEK(49200) or PEEK(-16366). The click is very brief, but a series of clicks will produce an audible buzz. Change lines 440 and 445 and run the program:

```
440 FOR L = 1 TO 50
445 X = PEEK(49200): NEXT L
```

Programming for Music

There is a way to produce a variety of musical notes and sound effects by using the Apple "click" sound. This is accomplished by clicking the Apple speaker at an extremely fast rate so that the resulting sound is no longer a buzz but a tone that can be adjusted with respect to duration and pitch. However, to do this a machine language subroutine must be used.

But a word of caution: my musical ability is minimal (and this is an overstatement). Because of this, I can show how to generate sound, and provide the necessary software, but you will need to exercise your own musical creativity to take full advantage of these techniques.

The machine language sound capability of the Apple is illustrated by the SOUND program shown below:

```
10 REM SOUND
20 REM
30 FOR K = 1 TO 21: READ A
40 POKE 801+K,A: NEXT K
50 DATA 174,32,3,173,48,192,136,208,5,206,
   33,3,240,6,202,208,245,76,34,3,96
60 HOME
70 GR
80 IF PEEK(-16287) > 127 THEN 80
90 IF PEEK(-16287) > 127 THEN 120
100 KY = PEEK(-16384): IF KY > 127 THEN 120
110 GOTO 90
120 IF KY = 155 THEN 230
130 POKE -16368,0
140 R = INT(RND(1)*40)
150 C = INT(RND(1)*15+1)
160 COLOR=C
170 GOSUB 5000
180 IF RND(1) > .5 THEN 210
190 HLIN 0,39 AT R
200 GOTO 80
210 VLIN 0,39 AT R
220 GOTO 80
230 TEXT: HOME
240 POKE -16368,0: END
5000 POKE 800,100
5010 POKE 801,200
5020 CALL 802
5030 POKE 800,75
5040 POKE 801,150
5050 CALL 802
5060 RETURN
```

Each time the switch is engaged, either a vertical or horizontal line having a randomly selected color is displayed. The sound that accompanies each switch response is produced by lines 5000 to 5050. These lines begin with the rather large line number so that this subroutine can be added to other BASIC programs that might be used.

The purpose of the machine language routine is the ability to click the speaker at an extremely rapid rate, a task that cannot be accomplished by using the slower running BASIC language. If entered as shown, the routine provides a reasonably efficient method for adding sound to BASIC programs.

To generate a note, three statements are used: one statement to signify the pitch or frequency, a second statement to indicate the duration of the note, and a third statement to CALL a special machine language subroutine (see lines 30, 40 and 50) to produce the sound. The frequency and duration of each note is determine by two POKE statements. The first value in each POKE indicates the memory location where the value is to be stored, and the second value is the frequency or duration value stored in the specified location.

Playing a Note

```
POKE 800,101      (sets pitch)
POKE 801,200      (sets duration)
CALL 802          (plays note)
```

To raise the pitch, the value poked in memory location 800 is deceased (the lowest value is 1); and to lower the pitch, this value is increased (the highest value is 255).

Changing Pitch

```
POKE 800,50       (high pitch)
POKE 800,150      (low pitch)
POKE 800,100      (average pitch)
```

After the pitch is set, a second POKE is used to indicate the duration. To decrease the duration, use a smaller value in this statement; and to increase the duration, use a larger value.

Changing Duration

```
POKE 801,50            (short duration)
POKE 801,200           (longer duration)
POKE 801,255           (maximum duration)
```

By changing the values in the subroutine, everything from music to sound effects can be created. Add these lines and see what sound is produced:

```
5000 POKE 800,101
5010 POKE 801,48
5020 CALL 802
5030 POKE 800,76
5040 POKE 801,48
5050 CALL 802
5060 POKE 800,60
5070 POKE 801,48
5080 CALL 802
5090 POKE 800,52
5100 POKE 801,96
5110 CALL 802
5120 POKE 800,60
5130 POKE 801.48
5140 CALL 802
5150 POKE 800,52
```

```
5160 POKE 801,255
5170 CALL 802
5180 RETURN
```

A different subroutine can be added by first deleting lines 5000 to 5170 and then adding the new sound routine:

```
5000 FOR K = 1 TO 200 STEP 10
5010 POKE 800,200-K
5020 POKE 801,10
5030 CALL 802
5040 NEXT K
```

The above subroutine generates a "snappy" run through the scale...although I'm not sure exactly what scale!

To produce a slightly different effect, make this change:

```
5010 POKE 800,10+K
```

Or make these changes to produce a type of computer mania:

```
5000 FOR K = 1 TO 25
5005 X = INT(RND(1)*245)
5010 POKE 800,10+X
```

Adding sound to a software application is a relatively easy matter, and the only limit is one's musical creativity (of which mine is extremely limited). To add a sound routine to another BASIC program, do the following:

1) Add the machine language program to the beginning of the program. Be careful not to enter the routine over line numbers that are currently in use;

2) Add the GOSUB statement in the program where sound created by the subroutine is generated;

3) Add the two POKE statements and CALL for each note in the music subroutine. Be sure not to enter the subroutine over existing program lines. If the routine is to be inserted in lines 5000 to 5180, LIST these lines to make sure they are empty.

Now back to the FACE program that began this chapter but was so void of musical enhancements. Load this program and add these lines:

```
22 FOR K = 1 TO 21: READ A
24 POKE 801 + K,A: NEXT K
26 DATA 174,32,3,173,48,192,136,208,5,206,
   33,3,240,6,202,208,245,76,34,3,96
550 X = X+10
560 POKE 800,250-X
570 POKE 801,100
580 CALL 802
590 RETURN
```

Each time the switch is engaged, a note is generated and a segment of the face displayed. Experiment with lines 560 and 570 to produce different sound/music effects.

```
560 POKE 800,150+X
```

or try

```
550 X = X+15
560 POKE 800,250-X
```

Variable X can also be set to a random number in line 550:

```
550 X = INT(RND(1)*200+50)
560 POKE 800,X
```

Or play a combination of notes following a switch response:

```
550 POKE 800,200: POKE 801,80: CALL 802
560 POKE 800,200: POKE 801,80: CALL 802
570 POKE 800,100: POKE 801,160: CALL 802
580
```

Two Switch Applications

Before leaving our "soundful" FACE program, let's consider the possibility of reading input from two switches. If the following modifications are added to the above program, alternate switch input, first from PEEK(-16287) then PEEK(-16286), must be used to display each face component.

```
520 IF SW = -16287 THEN SW = -16286: GOTO 525
521 SW = -16287
525 IF PEEK(SW) > 127 THEN 525
526 IF PEEK(SW) > 127 THEN 550
527 GOTO 526
```

How to Delete Program Sound

Although creating sound can be an important component of single switch programming, the ability to delete sound or turn off sound can be just as important. In some situations the sound used in a program might be disruptive to others in the class, unnecessary, or simply too long.

After the routine in the program used to generate the sound has been found, several approaches can be used to by-pass the unwanted feedback. If a subroutine is used to generate the sound, the subroutine ends with the word RETURN. By entering RETURN at the beginning of the subroutine, the sound routine can be effectively by-passed.

For the last program entered, add this statement and run the program:

```
5005 RETURN
```

A second method for by-passing sound, or anything else for that matter, is to "jump over" the routine with a GOTO statement. To illustrate how this works, first delete line 5005 so that the sound routine is intact. Next, add line 165 to jump over the GOSUB statement that calls the sound routine:

```
165 GOTO 180
```

Another technique that can be used is to input a value or code at the beginning of the program which determines whether or not sound is to be generated:

```
65 INPUT "DO YOU WANT SOUND (Y/N)? ";SD$
66 SD$ = LEFT$(SD$,1)
67 HOME
```

```
165 IF SD$ = "N" THEN 180
```

Yet another solution to the problem of a noisy computer is to connect headphones to the Apple or to the Echo™ or Echo IIb™ speech synthesizer (see the next section).

Synthesized Speech

This section describes how to generate Echo synthesized speech in conjunction with a single switch BASIC application. If an Echo is not available (and this is real "must" for anyone interested in single switch applications), the programs in this section will run but without corresponding speech output.

Echo Installation

In order to use an Echo speech synthesizer, first read the instructions in the Echo manual. Next, install the Echo card (be sure the system is off) into one of the slots (usually 4, 5 or 7). For IIGS users, the Echo cannot be installed in slot #3. If possible, use slot #7. If a IIGS is being used and slot #7 is not available, use the control panel to disable the assigned slot function.

The Echo board is installed by carefully inserting the board into one of the empty slots. To get the board into the slot, work the board into the slot by carefully "rocking" the board into position. After the speaker has been connected, the Echo should be ready to "synthesize."

For the Echo to be able to function, two programs must be on the DOS 3.3 disk: **TEXTALKER™** and **TT.OBJ** (short for TEXTALKER OBJECT program). Copy these programs from the Echo disk or program disk by using the DOS 3.3 System Master Disk, a copy program, or by first loading each file from the Echo or program disk and then saving each on the DOS 3.3 disk as shown below. The **B** preceding the LOAD, SAVE and RUN commands indicates that the files being used are binary.

```
BLOAD TEXTALKER                                  (Load from Echo disk)

BSAVE TEXTALKER, A37632, L528                    (Save on DOS 3.3 disk)

BLOAD TT.OBJ                                     (Load from Echo disk)

BSAVE TT.OBJ, A8192, L11956                      (Save on DOS 3.3 disk)
```

To show how the Echo can be turned on, used to say a word, enter the following:

```
BRUN TEXTALKER

W$ = "HELLO"
G$ = CHR$(5)
PRINT G$"T "W$" "G$"O"
PRINT W$
PRINT G$"B "W$" "G$"O"
PRINT W$
PRINT G$"B"
PRINT W$
```

So as not to confuse the Echo, it is a good idea to begin and end each string via the Echo with a space (i.e., " "W$" ").

Another simple routine that can be used to check and test the Echo speech is the following:

```
10 INPUT W$
20 PRINT CHR$(5)"T "W$" "CHR$(5)"O"
30 IF W$ = "Q" THEN END
40 GOT 10
```

Programming for Speech

The TALK program illustrates how use the Echo with a BASIC program. The first line in the program loads the TEXTALKER program (which, in turn, loads the TEXTALKER OBJECT program) necessary to generate speech.

```
10 REM TALK
20 REM
30 PRINT CHR$(4);"BRUN TEXTALKER"
40 FOR L = 1 TO 1000: NEXT L
50 G$ = CHR$(5): PRINT G$"O"
60 FOR K = 1 TO 8
70 READ S$(K),ES$(K)
80 NEXT K
90 DATA YES, YES
100 DATA NO, NO
110 DATA DRINK, I'M THIRSTY.
120 DATA MUSIC, I WANT TO LISTEN TO MUSIC.
130 DATA HELP, I NEED HELP!
140 DATA THANKS, THANK YOU!
150 DATA ?, I DON'T UNDERSTAND.
160 DATA !, THAT'S GREAT!
170 HOME
180 FOR D = 1 TO 8
190 VTAB D*2+2: HTAB 10
200 PRINT S$(D): NEXT D
210 VP = 1
220 VTAB VP*2+2: HTAB 10
230 INVERSE: PRINT S$(VP): NORMAL
240 FOR D = 1 TO 100
250 IF PEEK(-16287) > 127 THEN 320
260 IF PEEK(-16384) = 155 THEN 370
270 NEXT D
280 VTAB VP*2+2: HTAB 10
290 PRINT S$(VP)
300 VP = VP+1: IF VP > 8 THEN VP = 1
310 GOTO 220
320 VTAB 21: HTAB 3: PRINT ES$(VP)
330 PRINT G$"T "ES$(VP)" "G$"O"
340 FOR D = 1 TO 2000: NEXT D
350 VTAB 21: CALL -868
360 GOTO 180
370 POKE -16368,0
380 VTAB 23
390 END
```

The program first loads the software necessary to use the Echo card that has been installed. The CHR$(4) instruction in line 30 indicates that a DOS command follows (which is to load and run the binary file TEXTALKER from the program disk).

The program then reads eight pairs of words or statements, such that the first element of each pair is stored in S$(K) and the second element is stored in ES$(K). The first element of each pair is what is displayed and scanned on screen, and the second element is what is spoken by the Echo. For example, if the word DRINK is scanned and selected, the Echo generates I'M THIRSTY. The elements of each pair can be modified by changing the DATA statements which begin in line 90.

In line, 50 G$ is set to CHR$(5) or Control+E which is used to initiate all Echo commands. The following is a list of several important Echo commands (be sure to consult the Echo manual for additional commands that might be of use):

```
PRINT CHR$(5)"O"        (Print Only)
PRINT CHR$(5)"T"        (Talk Only)
PRINT CHR$(5)"B"        (Print and Talk)
```

If several changes are being made in a program which uses the Echo, a small inconvenience can occur each time the program is interrupted and then re-run. What happens is that the Echo is loaded each time the program is run, even though the program is already in memory. If using a disk that is entirely dedicated to Echo programs, this problem is solved by including the following statement in the HELLO program when the disk is first booted:

```
25 PRINT CHR$(4);"BRUN TEXTALKER"
```

Then change line 30 in the TALK program as follows:

```
30 PRINT CHR$(4);"PR#0"
```

This command reconnects the Echo following a program interruption. To see how this works interrupt the program, enter line 30, and then re-run the program. When the switch is engaged, the output is displayed on the screen but with no speech. Now enter PR#0, RUN the program, and the Echo is reconnected:

```
PR#0
RUN
```

If the Echo is talking but does not display input on the screen, enter the following:

```
PRINT CHR$(5);"B"
```

Before going on to the next program, modify one of the DATA statements in lines 90 to 160 in the TALK program as illustrated by the following:

```
120 DATA TV, I WANT TO WATCH TELEVISION.
```

In order to synthesize each stimulus statement as it is being scanned, add the following line:

```
235 PRINT G$"T "S$(VP)" "G$"O"
```

Internal Program Echo Reconnect

The Echo can be automatically reconnected from within a program whenever the application is run. This is accomplished by setting a memory location that is not used to a specified value. When the program is run, if the specified value is encountered, the Echo is reconnected by PR#0 rather than by re-running the TEXTALKER. The following changes illustrate how this modification is made using the TALK program described above.

```
25 IF PEEK(1000) = 99 THEN 37
35 POKE 1000,99
36 GOTO 40
37 PRINT CHR$(4);"PR#0"
```

When 99 (and this can be any value other than the value present when the Apple is first booted) is entered in location 1000, this value remains even if the CLEAR is used within a program, if the program is interrupted, or if a different program is run. The value 99 is erased when the Apple is booted.

To run the TEXTALKER program when the location 1000 has been set to a specified value (e.g., something in the program being used might have "bombed" the TEXTALKER), either eliminate the PEEK check statement or reset the location by entering:

```
POKE 1000,0
```

Echo Bugs

If the Echo TEXTALKER program has not been loaded, or the Echo has somehow been disconnected, and an attempt to generate speech has been made, the Echo commands and string is displayed but obviously without accompanying speech:

```
T HELLO O
```

When installing the Echo in a IIGS, use slot #7 if possible. If necessary, disable the normal slot setting by means of the control panel.

If another Apple utility program is used, especially one that uses an ampersand (&), this might "clobber" the Echo. If this happens, re-run, or if necessary, re-boot the system.

Echo Commands

To experiment with the various Echo commands available, load the TEXTALKER program:

```
BRUN TEXTALKER
```

Or if the TEXTALKER has already been loaded but is disconnected, reconnect the program by entering:

```
PR#0
```

Now use PRINT statements to elicit Echo speech:

```
PRINT "HELLO"

PRINT "PLEASE, PRESS THE SWITCH"
```

The Echo volume can be set by using **CONTROL+E N V** where N is a value between 0 and 15. The present or default value of volume is 12. This function can be used within a program or by a PRINT statement:

```
PRINT CHR$(5);9;"V"
```

or

```
PRINT CHR$(5)"9V"
```

To experiment with Echo commands, try following simple. Change the V to P in line 40
to vary the pitch rather than the volume.

```
10 PRINT CHR$(4)"PR#0"
20 INPUT X$
30 IF X$ = "0" THEN END
40 PRINT CHR$(5);X$;"V"
50 PRINT "HOW ARE YOU"
60 GOTO 10
```

The volume can vary from 0 (least loud) to 15 (most loud), and the default value when first run is 12; pitch can vary from 1 (lowest pitch) to 63 (highest pitch), and the default value when first run is 22. To add a delay between words, enter the following:

```
45 PRINT CHR$(5)"10D"
```
(the delay can range from 0 to 15)

When first run the default punctuation setting is SOME (**CONTROL+E"S" or PRINT CHR$(5)"S"**), the ALL punctuation setting is activated (e.g., the spacebar generates the word "space") by the following:

```
G$ = CHR$(5)
PR#0
PRINT CHR$(5)"A"
```

a MOST punctuation command is also possible: **PRINT G$"M"**.

When the Echo is first loaded, the TEXTALKER is set to say words. This is changed so that words are pronounced letter-by-letter by the following:

```
PRINT G$"L"
```

Word pronunciation is reset by

```
PRINT G$"W"
```

To change the speed for Echo speech to compressed or fast mode, enter

```
PRINT G$"C"
```

and this to change back to the default expanded or slow rate mode

```
PRINT G$"E"
```

The following brief can be used to test various Echo characteristics by generating alphabet characters A to Z. Change line 20, or add an additional line, to test additional Echo commands.

```
NEW
10 PRINT CHR$(4)"PR#0"
20 PRINT CHR$(5)"C"
20 FOR K = 1 TO 26
30 PRINT CHR$(64+K)
40 NEXT K
```

When the Echo is running, CONTROL+S can be used to momentarily interrupt the program or CONTROL+C to break the program run. For additional Echo command information, consult the Echo manual.

Phoneme Codes

When Control+V or CHR$(22) precedes a group of characters, the symbols are interpreted as phonemes rather than letters. The Echo IIb manual contains a list of the possible phoneme codes on Appendix E. To sample several phoneme codes words, be sure the Echo is connected (PR#0) and then set CV$ to CHR$(22):

```
CV$ = CHR$(22)
```

Now enter the following print statements so that each string os phoneme codes is preceded by CV$:

```
PRINT CV$"GQ3D"
PRINT CV$"S;R3Q"
PRINT CV$"KORE3KT"
PRINT CV$"HEL01"
PRINT CV$"SZ&"
```

In the first PRINT statement, the Q is the phoneme code for the long o as in book, and the value 3 following the vowel oo indicates that the vowel is stressed. The last PRINT statement is the name of the letter "Z". Although phonetically spelling words can be used in most instances to synthesize a word that has the desired pronunciation, the phoneme commands do add a degree of exactness to this "hit and miss" process.

Female Speech

The male robotic voice of the Echo can be replaced by female speech, but not without some difficulty and the loss of considerable text-to-speech flexibility. To experiment with this option, boot the Echo ProDOS IIb disk containing the Word Editor. Enter the words COMPUTER, ECHO II and HELLO. Use the closed-Apple+T combination to say a bracketed word: **(COMPUTER....)**.

Another way to experiment with the female voice capability is to BRUN the SAY program, load the sample word file, and then use the &SHOWWRDS to see what words are in this file. The &SAY command is then used to say the words in the word file:

```
BRUN SAY
BLODWRDS,"SAMPLE"
&SHOWWRDS
&SAY,"I","LIKE","THE","ECHO II"
```

Consult the Echo IIb manual for detailed information concerning the development and use of Fixed Vocabulary Female Speech. When done with ProDOS, remember to re-boot the DOS 3.3 program disk in order to use other programs described in this manual.

Matrix Scanning

Single Switch Spelling

The SPELLTALK program uses a matrix to scan the letters of the alphabet. The alphabet and punctuation are displayed using five screen lines. Each matrix row is scanned to first select a sequence of characters. A second scan is then used to select the specific letter or character in that sequence. Thus, to select a character two switch movements must be made: row selection and then the specific character selection.

Although row/element scanning is fairly efficient, he two switch task can be difficult for some individuals to conceptualize. The alternative, and a procedure better suited for smaller matrices, is sequential element scanning (see the language board section) in which each element of the matrix is scanned in sequential order.

A	**B**	**C**	**D**	**E**	**F**
G	**H**	**I**	**J**	**K**	**L**
M	**N**	**O**	**P**	**Q**	**R**
S	**T**	**U**	**V**	**W**	**X**
Y	**Z**	**.**	**?**		

After a character is selected, the letter is spoken and displayed at the bottom of the screen. To speak an entire word or message, press the T key. When entering this program, and when modifying lines 60 through 100, spacing is essential. There are three spaces between each character. In the program listing shown below, line 100 is entered as follows, where the symbol □ signifies a space:

```
100 S$(5)="Y□□□Z□□□.□□□?"
```

```
10 REM SPELLTALK
20 REM
30 PRINT CHR$(4);"BRUN TEXTALKER"
40 FOR L = 1 TO 1000: NEXT L
50 G$ = CHR$(5): PRINT G$"O"
60 S$(1) = "A   B   C   D   E   F"
70 S$(2) = "G   H   I   J   K   L"
80 S$(3) = "M   N   O   P   Q   R"
90 S$(4) = "S   T   U   V   W   X"
100 S$(5) = "Y   Z   .   ?"
110 HOME
120 HTAB 15: PRINT "SPELLTALK"
130 VP = 1
140 FOR D = 1 TO 5
150 VTAB VP+D*3: HTAB 9
160 PRINT S$(D): NEXT D
170 VTAB VP*3+1: HTAB 9
180 INVERSE: PRINT S$(VP): NORMAL
190 GOSUB 510
200 IF SW = 1 THEN SW = 0: GOTO 270
210 VTAB VP*3+1: HTAB 9
220 PRINT S$(VP)
230 VP = VP+1: IF VP > 5 THEN VP = 1
240 VTAB VP*3+1: HTAB 9
250 INVERSE: PRINT S$(VP): NORMAL
260 GOTO 190
270 VTAB VP*3+1: HTAB 9
280 PRINT S$(VP)
290 VTAB VP*3+1: HTAB 9
300 INVERSE
310 PRINT MID$(S$(VP),1,1)
320 NORMAL
330 HP = 9
340 GOSUB 510
```

```
350 VTAB VP*3+1: HTAB HP
360 PRINT MID$(S$(VP),HP-8,1)
370 IF SW = 1 THEN 460
380 HP = HP+4
390 IF HP > 29 THEN HP = 9
400 VTAB VP*3+1: HTAB HP
410 INVERSE
420 PRINT MID$(S$(VP),HP-8,1)
430 NORMAL
440 GOTO 340
450 SW = 1
460 C$ = MID$(S$(VP),HP-8,1)
470 M$ = M$ + C$
480 PRINT G$"T "C$" "G$"O"
490 VTAB 20: HTAB 1: PRINT M$
500 GOTO 130
510 IF PEEK(-16287) > 127 THEN 510
520 FOR D = 1 TO 100
530 IF PEEK(-16287) > 127 THEN 560
540 KY = PEEK(-16384): IF KY > 127 THEN 560
550 NEXT D: SW = 0: RETURN
560 POKE -16368,0: SW = 1
570 IF KY = 155 THEN 680
580 IF KY = 212 THEN 600
590 GOTO 620
600 PRINT G$"T "M$" "G$"O"
610 GOTO 130
620 IF KY = 195 THEN 640
630 RETURN
640 M$ = ""
650 VTAB 20: HTAB 1: CALL -868
660 SW = 0
670 RETURN
680 END
```

The SPELLTALK program can be modified to say each word or message by changing line 100 and then adding line 465. As shown, the / symbol indicates that the word or message displayed is to be spoken. In line 465, if the symbol selected is /, control is sent to line 600 and the word or message in M$ is spoken. When entering line 100 be sure that there are three spaces between each character:

```
100 S$(5) = "Y   Z   .   ?   /"
465 IF C$ = "/" THEN 600
```

The message displayed at the bottom of the screen is cleared by pressing the C key. However, this can also be switch controlled by selecting a keyboard character, other than one that is used to display a message, modifying line 100, and then adding line 466. In this modification, the asterisk "*" signifies that the message at the bottom of the screen is to be cleared:

```
100 S$(5) = "Y   Z   .   ?   /   *"
466 IF C$ = "*" THEN 640
665 IF C$ = "*" THEN 130
```

To decrease or increase the scan speed, decrease or increase the value in the loop contained in line 520:

```
520 FOR D = 1 TO 50  (faster scan speed)
```

or

```
520 FOR D = 1 TO 50  (slower scan speed)
```

Of course an INPUT statement can always be used to specify scan speed:

```
110 HOME
120 HTAB 15: PRINT "SPELLTALK"
121 VTAB 7
122 INPUT "SCAN SPEED (1=FAST, 9=SLOW): ";ST$
123 ST = VAL(ST$): IF ST = 0 THEN ST = 5
124 VTAB 4: CALL -958
520 FOR D = 1 TO 10*ST
```

For students who are visually impaired or simply to provide additional auditory cues as each row letter is scanned, the following modification presents each row letter via the Echo as it is scanned:

```
345 L$ = MID$(S$(VP),HP-8,1)
346 PRINT G$"T "L$" "G$"O"
```

Because matrix scanning requires two switch movements to select a character, scan speed becomes an extremely important factor. The goal should be to select a speed that allows the student sufficient time to select characters, but a scan time that is not too slow in view of the student's switch response speed. Regardless of the type of matrix scan program being used, the students ability to use the matrix should be monitored in order adjust the scan speed when necessary.

If the two switch task is not understood, or as a way to introduce the complete letter matrix, the first two lines of the matrix can be displayed by the following changes:

```
140 FOR D = 1 TO 2
230 VP = VP+1: IF VP > 2 THEN VP = 1
```

Using only the first two lines of the matrix, the student can be directed to spell words such as DAD, BIG, HI, FACE, etc. An even simpler modification is to present only the first line of the matrix and then spell words such as BE, ADD,DAD, FACE, etc.

```
140 FOR D = 1 TO 1
230 VP = VP+1: IF VP > 1 THEN VP = 1
```

The above two modifications could also be enhanced by using frequently occurring letters in the matrix rows being scanned. See the Letter Prediction modification discussed below.

A sixth matrix line can be added if additional commands are necessary. The SPELLTALK-2 program listing contains all of the above modifications and a sixth matrix line. The first element of the last row can be used to insert a space, and the second element of the row is used to erase the last letter entered. The remaining row elements provide additional symbols and punctuation that can be used.

```
10 REM SPELLTALK-2
20 REM
30 PRINT CHR$(4);"BRUN TEXTALKER"
40 FOR L = 1 TO 1000: NEXT L
50 G$ = CHR$(5): PRINT G$"O"
60 S$(1) = "A    B    C    D    E    F"
70 S$(2) = "G    H    I    J    K    L"
80 S$(3) = "M    N    O    P    Q    R"
90 S$(4) = "S    T    U    V    W    X"
```

```
100 S$(5) = "Y   Z   .   ?   /   *"
105 S$(6) = "    <   !   ,   =   +"
110 HOME
120 HTAB 15: PRINT "SPELLTALK"
121 VTAB 7
122 INPUT "SCAN SPEED (1=FAST, 9=SLOW): ";ST$
123 ST = VAL(ST$): IF ST = 0 THEN ST = 5
124 VTAB 4: CALL -958
130 VP = 1
140 FOR D = 1 TO 6
150 VTAB VP+D*3: HTAB 9
160 PRINT S$(D): NEXT D
170 VTAB VP*3+1: HTAB 9
180 INVERSE: PRINT S$(VP): NORMAL
190 GOSUB 510
200 IF SW = 1 THEN SW = 0: GOTO 270
210 VTAB VP*3+1: HTAB 9
220 PRINT S$(VP)
230 VP = VP+1: IF VP > 6 THEN VP = 1
240 VTAB VP*3+1: HTAB 9
250 INVERSE: PRINT S$(VP): NORMAL
260 GOTO 190
270 VTAB VP*3+1: HTAB 9
280 PRINT S$(VP)
290 VTAB VP*3+1: HTAB 9
300 INVERSE
310 PRINT MID$(S$(VP),1,1)
320 NORMAL
330 HP = 9
340 GOSUB 510
350 VTAB VP*3+1: HTAB HP
360 PRINT MID$(S$(VP),HP-8,1)
370 IF SW = 1 THEN 460
380 HP = HP+4
390 IF HP > 29 THEN HP = 9
400 VTAB VP*3+1: HTAB HP
410 INVERSE
420 PRINT MID$(S$(VP),HP-8,1)
430 NORMAL
440 GOTO 340
450 SW = 1
460 C$ = MID$(S$(VP),HP-8,1)
461 IF C$ = "/" THEN 600
462 IF C$ = "*" THEN 640
463 IF C$ < > "<" THEN 470
464 IF LEN(M$) = 1 THEN M$ = "": GOTO 466
465 M$=LEFT$(M$, LEN(M$)-1)
466 VTAB 20: HTAB 1: CALL -868
467 GOTO 490
468 IF LEN(M$) = 1 THEN M$ = "": GOTO 490
469 M$ = LEFT$(M$,LEN(M$)-1): GOTO 490
470 M$ = M$ + C$
480 PRINT G$"T "C$" "G$"O"
490 VTAB 20: HTAB 1: PRINT M$
500 GOTO 130
510 IF PEEK(-16287) > 127 THEN 510
520 FOR D = 1 TO 10*ST
530 IF PEEK(-16287) > 127 THEN 560
540 KY = PEEK(-16384): IF KY > 127 THEN 560
550 NEXT D: SW = 0: RETURN
560 POKE -16368,0: SW = 1
570 IF KY = 155 THEN 680
```

```
580 IF KY = 212 THEN 600
590 GOTO 620
600 PRINT G$"T "M$" "G$"O"
610 GOTO 130
620 IF KY = 195 THEN 640
630 RETURN
640 M$ = ""
650 VTAB 20: HTAB 1: CALL -868
660 SW = 0
665 IF C$ = "*" THEN 130
670 RETURN
680 END
```

Letter Prediction

The letters for the SPELLTALK programs can be arranged sequentially, in a QWERTY or traditional keyboard manner, or by letter frequency. The following changes result in words being scanned from the most to least frequently appearing letters:

```
60 S$(1) = "E    T    A    O    I    N"
70 S$(2) = "S    H    R    D    L    C"
80 S$(3) = "U    M    W    F    G    Y"
90 S$(4) = "P    B    V    K    J    X"
100 S$(5) = "Q    Z    .    ?    !     "
```

Language Boards

A 3X3 Language Board Matrix

The TALK and SPELLTALK program can be combined to produce a language board. The 3X3 (three rows by three columns) language board shown below scans a matrix comprised of three rows and three columns. Matrix element scanning is sequential and occurs in the following order:

```
1  2  3
4  5  6
7  8  9
```

A screen overlay is used to signify what each matrix element represents. When a particular matrix element is scanned, the Echo message corresponding to that element is displayed.

Line 360 determines the vertical and horizontal screen position of each cursor scan, and line 360 finds the Echo statement that corresponds to this screen position. If the switch is engaged, the values of HX and VX which correspond to H and V are found, and the contents of EM$(VX,HX) are presented via the Echo (line 380).

The Echo entries corresponding to the matrix elements are contained in DATA statements, beginning in line 490. The DATA entries can be words, phrases, or a whatever alphanumeric string that is appropriate as long as the string is no more than 255 characters.

```
10 REM TALKBOARD-3X3
20 REM
30 DIM EM$(6,6)
40 HOME
50 PRINT CHR$(4);"BRUN TEXTALKER"
60 for L = 1 TO 1000: NEXT L
```

```
70 G$=CHR$(5): PRINT G$"O"
80 HOME: HTAB 12
90 PRINT "TALKBOARD-3X3"
100 FOR K = 1 TO 8: E$ = E$ + CHR$(32): NEXT K
110 FOR K = 1 TO 21: READ A
120 POKE 801+K,A: NEXT K
130 DATA 174,32,3,173,48,192,136,208,5,206,
    33,3,240,6,202,208,245,76,34,3,96
140 VTAB 8
150 INPUT "SCAN SPEED (1=FAST TO 9=SLOW): ";ST$
160 ST = VAL(ST$): IF ST < 1 THEN ST = 5
170 FOR J = 1 TO 3
180 FOR K = 1 TO 3
190 READ EM$(J,K)
200 NEXT K: NEXT J
210 V = 1: H = 1
220 HOME
230 INVERSE
240 FOR K = 0 TO 6
250 VTAB V+K: HTAB H
260 PRINT E$: NEXT K: NORMAL
270 IF PEEK(-16287) > 127 THEN 270
280 POKE 800,150: POKE 801,100: CALL 802
290 FOR D = 1 TO ST*10
300 IF PEEK(-16287) > 127 THEN 340
310 KY = PEEK(-16384): IF KY > 127 THEN 350
320 NEXT D
330 H = H+15: IF H > 31 THEN H = 1: V = V+7:
    IF V > 15 THEN V = 1
340 GOTO 210
350 POKE -16368,0
360 HX = (H-1)/15+1: VX = (V-1)/7+1
370 VTAB 21: HTAB 1
380 PRINT G$"T "EM$(VX,HX)" "G$"O"
390 VTAB 23: HTAB 5
400 PRINT "(PRESS KEY OR SWITCH TO CONTINUE)"
410 IF PEEK(-16287) > 127 THEN 410
420 IF PEEK(-16287) > 127 THEN 210
430 KY = PEEK(-16384): IF KY > 127 THEN 450
440 GOTO 420
450 POKE -16368,0
460 IF KY = 155 THEN 480
470 GOTO 210
480 END
490 DATA HELLO!
500 DATA I AM HUNGRY.
510 DATA I AM THIRSTY.
520 DATA I LIKE THAT.
530 DATA I DON'T UNDERSTAND.
540 DATA YES
550 DATA NO
560 DATA CAN I WATCH TV?
570 DATA WILL YOU READ TO ME?
```

By saving the program using different file names, a great many useful language experience boards can be created. For example, the 3X3 board can be used with an overlay comprised of 9 different colors and having the student select the appropriate matrix element in response to a verbal prompt (e.g., "Which is blue?"). Or a board can be created that displays eating utensils, pictures of friends and family members, activities, etc.

4X4 Language Board Matrix

The language board can be expanded to a 4X4 matrix by making these modifications:

```
10 REM TALKBOARD-4X4
90 PRINT "TALKBOARD-4X4"
100 FOR K = 1 TO 6: E$ = E$ + CHR$(32): NEXT K
170 FOR J = 1 TO 4
180 FOR K = 1 TO 4
240 FOR K = 0 TO 4
330 H = H+11: IF H > 34 THEN H = 1: V = V+5:
    IF V > 16 THEN V = 1
360 HX = (H-1)/11+1: VX = (V-1)/5+1
490 DATA HELLO!
500 DATA I AM HUNGRY.
510 DATA I AM THIRSTY.
520 DATA I LIKE THAT.
530 DATA I DON'T UNDERSTAND.
540 DATA YES
550 DATA NO
560 DATA CAN I WATCH TV?
570 DATA WILL YOU READ TO ME?
580 DATA WHAT TIME IS IT?
590 DATA I'M TIRED.
600 DATA THAT IS CORRECT.
610 DATA THAT IS NOT CORRECT.
620 DATA MY NAME IS FRED.
630 DATA WHAT IS THAT?
640 DATA THAT'S FUNNY!
```

A 5X5 Language Board Matrix

Before using the following 5X5 matrix, be sure to add 25 DATA statements to replace the TALKBOARD-5X5 file, beginning in line 490.

```
10 REM TALKBOARD-5X5
90 PRINT "TALKBOARD-5X5"
100 FOR K = 1 TO 6: E$ = E$ + CHR$(32): NEXT K
170 FOR J = 1 TO 5
180 FOR K = 1 TO 5
240 FOR K = 0 TO 3
330 H = H+8: IF H > 32 THEN H = 1: V = V+4:
    IF V > 20 THEN V = 1
360 HX = (H-1)/8+1: VX = (V-1)/4+1
490 DATA A
.
.
.
730 DATA Y
```

2X2 Language Board Matrix

Because of the need to develop flexible language systems, there is a tendency to create talkboards that are too complex for the individual's language system. The following two modifications illustrate how the program described above can be reduced to a 2X2 language matrix or to a simple binary language task. For the binary task, an overlay is used to depict two items. When the switch is activated, the Echo presents the label/message corresponding to the overlay component:

Because the words presented by the Echo are not displayed on screen, the DATA statement entries can be entered phonetically. In the case of the word SECOND (see line 500 in the below listing), the spelling SECUND provides a better Echo pronunciation.

The screen is divided into four quadrants and each quadrant is scanned until the switch is engaged. Each quadrant corresponds to a DATA statement entry, beginning in line 490. The DATA statement entries can be either a single words or phrases. Although strings can be up to 255 alphanumeric characters in length, simple Echo entries are most appropriate for a simplified 2X2 language board screen display

```
10 REM TALKBOARD-2X2
90 PRINT "TALKBOARD-2X2"
100 FOR K = 1 TO 18: E$ = E$+CHR$(32): NEXT K
170 FOR J = 1 TO 2
180 FOR K = 1 TO 2
240 FOR K = 0 TO 8
330 H = H+20: I H > 21 THEN H = 1:
    V = V+12: IF V > 13 THEN V = 1
360 HX = (H-1)/20+1: VX = (V-1)/12+1
490 DATA FIRST
500 DATA SECUND
510 DATA THIRD
520 DATA FOURTH
```

Touchwindow Matrix Input

All of the language boards discussed in this section can be modified for use with a Touchwindow. However, small matrix entries can result in a task that requires considerable motor skills. Because of this, the 2X2 matrix, and the binary language board discussed below, provide the best opportunity for using Touchwindow input with a language board.

To use a Touchwindow with a 2X2 matrix, change line 300 which reads the input, and then add lines 360, 361 and 362 to set the coordinates VX and HX to correspond to the Touchwindow input:

```
300 IF PDL(0) > 10 OR PDL(1) > 10 THEN 350
360 VX=1: HX=1
361 IF PDL(0) > 125 THEN HX = 2
362 IF PDL(1) > 125 THEN VX = 2
```

Binary Language Board

This represents the simplest language board scheme. The screen is divided into two halves. A two item overlay is used, and each overlay component has a corresponding Echo entry. This format can be used to teach the language board concept. For example, the overlay might contain a picture of the student and another person in the room. When the switch is engaged, either the name of the student or the other person is presented via the Echo.

```
10 REM TALKBOARD-BINARY
90 PRINT "TALKBOARD-BINARY"
100 FOR K = 1 TO 15: E$ = E$+CHR$(32): NEXT K
170 FOR J = 1 TO 1
180 FOR K = 1 TO 2
240 FOR K = 2 TO 19
330 H = H+20: IF H > 22 THEN H = 1
360 HX = (H-2)/20+1: VX = 1
```

```
490 DATA FIRST
500 DATA SECUND
```

To use the Touchwindow routine with the binary language board discussed below, VX in line 360 would be set to 1.

```
300 IF PDL(1) > 10 THEN 350
360 VX = 1: HX = 1
361 IF PDL(0) > 125 THEN HX = 2
```

Talkboard Graphics

Instead of using a null string to display each scan block, the display can be in graphics mode. However, when using ordinary low-resolution graphics the display tends to be somewhat slow and the result is not as crisp as the above procedure. This problem can be circumvented by using a machine language routine, but there is a corresponding increase in the program complexity and a decrease in programming flexibility.

The following modifications illustrates how the TALKBOARD-2X2 program is modified to use simple low-resolution graphics. The primary advantage of this program is the ability to modify the scan color in line 226; while the disadvantage is the somewhat slow scan display without resorting to a machine language routine.

```
10 REM TALKBOARD-GRAPHICS
90 PRINT "TALKBOARD-GRAPHICS"
100
225 GR
226 COLOR=2
240 FOR K = 4 TO 15
250 VLIN 4+V,15+V AT K+H
260 NEXT K
330 H = H+20: IF H > 21 THEN H = 1:
    V = V+20: IF V > 21 THEN V = 1
360 HX = (H-1)/20+1: VX = (V-1)/20+1
```

Machine Language Graphics

The advantage of a machine language routine to produce a scan is speed. If at all possible, an almost instantaneous screen scan is desirable. Unfortunately, the advantages of added program speed through machine language programming also limits the ease of making changes in the BASIC code.

The following program illustrates how a machine language subroutine is used to draw in low-resolution graphics. This program might be of interest to those requiring added program speed or with an interest in advanced programming applications. The TALKBOARD-MG program also contains a component (described above) to automatically reconnect the Echo after the program has been run and the interrupted. Line 160 contains the machine language code for producing sound; and line 190 contains the machine language code for generating the screen cursor used to highlight language board entries.

```
10 REM TALKBOARD-MG
20 REM
30 DIM EM$(6,6)
40 HOME
50 IF PEEK(1000) = 99 THEN 90
60 PRINT CHR$(4);"BRUN TEXTALKER"
60 for L = 1 TO 1000: NEXT L
```

```
70 POKE 1000,99
80 GOTO 100
90 PRINT CHR$(4);"PR#0"
100 FOR L = 1 TO 1000: NEXT L
110 G$=CHR$(5): PRINT G$"O"
120 HOME: HTAB 8
130 PRINT "TALKBOARD-MACHINE GRAPHICS"
140 FOR K = 1 TO 21: READ A
150 POKE 801+K,A: NEXT K
160 DATA 174,32,3,173,48,192,136,208,5,206,
    33,3,240,6,202,208,245,76,34,3,96
170 FOR K = 1 TO 17: READ X
180 POKE 900+K,X: NEXT K
190 DATA 160,0,169,32,133,45,169,20,32,40,248,
    200,192,20,208,242,96
200 VTAB 8
210 INPUT "SCAN SPEED (1=FAST TO 9=SLOW): ";ST$
220 ST = VAL(ST$): IF ST < 1 THEN ST = 5
230 FOR J = 1 TO 1
240 FOR K = 1 TO 3
250 READ EM$(J,K)
260 NEXT K: NEXT J
270 RS = 2: RE = 37
280 CS = 2
290 GR: HOME
300 COLOR=15
310 GOSUB 330
320 GOTO 370
330 POKE 908,RS: POKE 904,RE
340 POKE 902,CS: POKE 914,CS+1
350 CALL 901
360 RETURN
370 IF PEEK(-16287) > 127 THEN 370
380 POKE 800,150: POKE 801,100: CALL 802
390 FOR D = 1 TO ST*10
400 IF PEEK(-16287) > 127 THEN 480
410 KY = PEEK(-16384): IF KY > 127 THEN 480
420 NEXT D
430 COLOR=0
440 GOSUB 330
450 CS = CS + 18: IF CS > 21 THEN CS = 2
460 FOR L'= 1 TO 1000: NEXT L
470 GOTO 300
480 POKE -16368,0
490 HX = (CS-2)/18+1: VX = 1
500 VTAB 21: HTAB 1
510 PRINT G$"T "EM$(VX,HX)" "G$"O"
520 VTAB 23: HTAB 5
530 PRINT "(PRESS KEY OR SWITCH TO CONTINUE)"
540 IF PEEK(-16287) > 127 THEN 540
550 IF PEEK(-16287) > 127 THEN 270
560 KY = PEEK(-16384): IF KY > 127 THEN 580
570 GOTO 550
580 POKE -16368,0
590 IF KY = 155 THEN 610
600 GOTO 270
610 TEXT: HOME
620 END
630 DATA GLASS
640 DATA HOUSE
```

In line 270, RS signifies the beginning screen row (a value between 0 and

39) and RE the ending screen row. The value RE must be equal to or greater than RS. The value CS signifies the beginning column row. The machine language subroutine is called in line 350. For those interested in advanced programming, or just to see what machine language code looks line, enter CALL -151 to access the Apple monitor which runs machine language programs. Next, enter 385L after the asterisk to list the machine language subroutine:

```
CALL -151
*385L
```

use Control+Reset or **FP** to exit the monitor.

The machine language subroutine used to produce the sound can be listed by:

```
CALL -151
*322L
```

Probability Language Sequencing

A modification which is appropriate for every type of language board activity, especially when sequential matrix element scanning is used, is to order the language concepts in terms of usage (also see the Letter Prediction modification used with the SPELLTALK-2 program). Initially, the language concepts comprising the matrix can be entered based on previous experience or a "best guess" estimate. With experience, and based on actual usage, language concepts can be arranged in the matrix based on the probability of being selected.

Expressive Language Board Applications

Single switch reading applications are discussed extensively in Chapter 8. The following applications are presented because of the use of matrix scanning and Echo output.

Row/Element Matrix Scanning

The READTALK program uses a 5X5 matrix scan procedure to select from up to 25 different Echo responses. If the matrix element MUSIC is selected, the Echo produces I WANT TO LISTEN TO MUSIC; and if the ? is selected, the Echo produces I DON'T UNDERSTAND.

Unlike the 5X5 language board matrix described above, READTALK uses a row/element scanning procedure. Each row is scanned in sequential order such that the entire row is highlighted. When the switch is engaged, each row element of the designated row is scanned in sequential order. When the switch is engaged a second time, the highlighted row element becomes the selected matrix entry.

```
10 REM READTALK
20 REM
30 DIM M$(6,6),EM$(6,6)
40 ST = 5
50 PRINT CHR$(4);"BRUN TEXTALKER"
60 G$=CHR$(5): PRINT G$"O"
70 FOR K = 1 TO 40: E$ = E$ + CHR$(32): NEXT K
80 FOR K = 1 TO 40: H$ = H$ + "-": NEXT K
90 FOR J = 1 TO 5
```

```
100 FOR K = 1 TO 5
110 READ M$(J,K),EM$(J,K)
120 LT = 7: IF K = 5 THEN LT = 9
130 S$(J) = S$(J) + LEFT$(M$(J,K) + E$,LT)
140 NEXT K: NEXT J
150 HOME: HTAB 16: PRINT "READTALK"
160 VTAB 2: HTAB 1: PRINT H$
170 VTAB 19: HTAB 1: PRINT H$
180 FOR D = 1 TO 5
190 VTAB 1+D*3: HTAB 1
200 PRINT S$(D): NEXT D
210 VP = 1
220 VTAB VP*3+1: HTAB 1
230 INVERSE: PRINT S$(VP): NORMAL
240 GOSUB 530
250 IF SW = 1 THEN SW = 0: GOTO 320
260 VTAB VP*3+1: HTAB 1
270 PRINT S$(VP)
280 VP = VP+1: IF VP > 5 THEN VP = 1
290 VTAB VP*3+1: HTAB 1
300 INVERSE: PRINT S$(VP): NORMAL
310 GOTO 240
320 VTAB VP*3+1: HTAB 1
330 PRINT S$(VP)
340 VTAB VP*3+1: HTAB 1
350 INVERSE: PRINT M$(VP,1): NORMAL
360 HP = 1
370 GOSUB 530
380 VTAB VP*3+1: HTAB HP*7-6
390 PRINT M$(VP,HP)
400 IF SW = 1 THEN 460
410 HP = HP+1: IF HP > 5 THEN HP = 1
420 VTAB VP*3+2: HTAB HP*7-6: PRINT
430 VTAB VP*3+1: HTAB HP*7-6
440 INVERSE: PRINT M$(VP,HP): NORMAL
450 GOTO 370
460 SW = 1
470 VTAB 21: HTAB 1: CALL -868
480 VTAB 21: HTAB 3: PRINT EM$(VP,HP)
490 PRINT G$"T "EM$(VP,HP)" "G$"O"
500 FOR L = 1 TO 1500: NEXT L
510 VTAB 21: CALL -958
520 GOTO 210
530 FOR D = 1 TO ST*10
540 IF PEEK(-16287) > 127 THEN 570
550 KY = PEEK(-16384): IF KY > 127 THEN 570
560 NEXT D: SW = 0: RETURN
570 POKE -16368,0: SW = 1
580 FOR L = 1 TO 500: NEXT L
590 IF KY = 155 THEN 610
600 RETURN
610 END
620 REM
630 REM DATA STATEMENTS
640 REM
650 DATA YES,YES
660 DATA NO,NO
670 DATA HI, HELLO! HOW ARE YOU?
680 DATA SORRY, I'M SORRY.
690 DATA GOODBYE,GOODBYE
700 DATA DRINK, I'M THIRSTY.
710 DATA FOOD, I'M HUNGRY.
```

```
720 DATA SNACK, CAN I HAVE A SNACK?
730 DATA TIRED, I'M TIRED.
740 DATA BATHROOM, I NEED TO USE THE BATHROOM.
750 DATA TV, CAN I WATCH TV?
760 DATA RADIO, CAN I LISTEN TO THE RADIO?
770 DATA MUSIC, I WANT TO LISTEN TO MUSIC.
780 DATA PLAY, I WANT TO PLAY.
790 DATA NINTENDO, CAN I PLAY NINTENDO?
800 DATA HELP, I NEED HELP?
810 DATA STOP, "STOP, PLEASE!"
820 DATA OUCH!, SOMETHING HURTS!
830 DATA BORED, I'M BORED.
840 DATA CAN'T, I CAN'T DO IT.
850 DATA EXCUSE, "EXCUSE ME, PLEASE."
860 DATA THANKS, THANK YOU!
870 DATA FUNNY, THAT IS VERY FUNNY.
880 DATA ?, I DON'T UNDERSTAND.
890 DATA !, THAT'S GREAT!
```

The above program should be individualized so that the words/statements best meet specific student learning needs. Because of this, it might be easier to specify the scan speed in the program (e.g., line 40) rather than entering the speed each time the program is run. If this is not the case, enter a scan speed input routine as shown in one of the TALKBOARD programs.

The matrix elements for READTALK are contained in DATA statements beginning in line 650. For each DATA statement, the first entry is the matrix element displayed on screen and the second entry is the corresponding statement presented via the Echo. For example, line 890 can be changed as follows:

```
890 DATA WOW, THAT'S TERRIFIC!
```

Expressive Language Boards

The READTALK program can be modified to present a series of frequently used words so as to create an expressive language activity entailing word selection and syntax. The WORDTALK program listed below is definitely limited with respect to the number of words and the length of each word that can be used. However, the program can be used to provide a great many different expressive language activities, and an individual who can successfully use this program will have demonstrated readiness for a more comprehensive expressive language board system.

The entries for the first four elements of each row cannot exceed seven characters, while the last entry of each row can be up to nine characters in length. Lines 120 and 130 insure that these limits are not exceeded.

```
10 REM WORDTALK
20 REM
30 DIM M$(6,6),EM$(6,6)
40 ST = 3
50 PRINT CHR$(4);"BRUN TEXTALKER"
60 G$ = CHR$(5): PRINT G$"O"
70 FOR K = 1 TO 40: E$ = E$ + CHR$(32): NEXT K
80 FOR K = 1 TO 40: H$ = H$ + "-": NEXT K
90 FOR J = 1 TO 5
100 FOR K = 1 TO 5
110 READ M$(J,K): M$(J,K) = M$(J,K)
120 LT = 7: IF K = 5 THEN LT = 9
130 S$(J) = S$(J) + LEFT$(M$(J,K) + E$,LT)
```

```
140 NEXT K: NEXT J
150 HOME: HTAB 16: PRINT "WORDTALK"
160 VTAB 2: HTAB 1: PRINT H$
170 VTAB 19: HTAB 1: PRINT H$
180 FOR D = 1 TO 5
190 VTAB 1+D*3: HTAB 1
200 PRINT S$(D): NEXT D
210 VP = 1
220 VTAB VP*3+1: HTAB 1
230 INVERSE: PRINT S$(VP): NORMAL
240 GOSUB 580
250 IF SW = 1 THEN SW = 0: GOTO 320
260 VTAB VP*3+1: HTAB 1
270 PRINT S$(VP)
280 VP = VP+1: IF VP > 5 THEN VP = 1
290 VTAB VP*3+1: HTAB 1
300 INVERSE: PRINT S$(VP): NORMAL
310 GOTO 240
320 VTAB VP*3+1: HTAB 1
330 PRINT S$(VP)
340 VTAB VP*3+1: HTAB 1
350 INVERSE: PRINT M$(VP,1): NORMAL
360 HP = 1
370 GOSUB 580
380 VTAB VP*3+1: HTAB HP*7-6
390 PRINT M$(VP,HP)
400 IF SW = 1 THEN 460
410 HP = HP+1: IF HP > 5 THEN HP = 1
420 VTAB VP*3+2: HTAB HP*7-6: PRINT
430 VTAB VP*3+1: HTAB HP*7-6
440 INVERSE: PRINT M$(VP,HP): NORMAL
450 GOTO 370
460 SW = 1
470 W$ = EM$(VP,HP)
480 IF W$ = "*CLEAR" THEN M$ = "": GOTO 540
490 IF W$ < > "*TALK" THEN 530
500 VTAB 21: HTAB 1
510 PRINT G$"T "M$" "G$"O"
520 GOTO 210
530 M$ = M$+W$+" "
540 VTAB 21: HTAB 1: CALL -868
550 VTAB 21: HTAB 1: PRINT M$
560 FOR L = 1 TO 1500: NEXT L
570 GOTO 210
580 FOR D = 1 TO ST*10
590 IF PEEK(-16287) > 127 THEN 620
600 KY = PEEK(-16384): IF KY > 127 THEN 620
610 NEXT D: SW = 0: RETURN
620 POKE -16368,0: SW = 1
630 FOR L = 1 TO 500: NEXT L
640 IF KY = 155 THEN 660
650 RETURN
660 END
670 REM
680 REM DATA STATEMENTS
690 REM
700 DATA I,WAS,IS,HE,THAT
710 DATA RIGHT,WRONG,WHAT,THIS,HUNGRY
720 DATA HE,SHE,FUNNY,AM,PLAYING
730 DATA TO,PLAY,READ,SCHOOL,HOME
740 DATA A,TV,GAME,*TALK,*CLEAR
```

The last two matrix entries (see line 740) are used to present the commands used to initiate Echo speech (***TALK**), and to clear the screen (***CLEAR**). When either of these instructions are encountered (lines 480 and 490), the Echo is activated or the screen is cleared. The DATA statements beginning in line 700 can be changed to present words from a story, specific content words, or language structures (e.g., the use of adjectives).

The message produced by WORDTALK is presented via the echo by selecting *TALK in row five. To present each word through the Echo each time a matrix element is selected, add line 545:

```
545 PRINT G$"T "W$" "G$"O"
```

Inverse Lowercase

The Apple does not have an inverse mode for lower-case characters. However, if an 80-column card is installed in the Apple being used, activate the card and then set the display to 40-columns. Because the 80-Column Text Card does have an inverse mode for lower-case characters, lower-case characters in inverse mode will look like real words and not a message form Mars.

```
45 PRINT CHR$(4);"PR#3"
46 PRINT CHR$(17)
```

Note: For Apple II+ users, lowercase characters appear as meaningless when displayed on screen. For example, the word **yes** appears as **9%3**. If an 80-column card is installed, the card can certainly make lowercase characters meaningful. However, consult the manual for the 80-column card being used for necessary instructions to display characters and clear the screen.

80-Column Text Card Display

Most single switch applications are best suited for the usual 40-column screen display. However, for older students, adults, and for students able to use a small typeset, an 80-column display can be an important enhancement.

The 80-Column card is inserted in slot #3 so the program is modified to first activate the card by adding line 45. Many of the changes are minor such as changing the HTAB setting from 1 to 2. The major change entails the use of the HTAB when going beyond 40 columns in that HTAB doesn't work beyond this point. To tab columns 41 to 81, a POKE must be used (see line 380). The CHR$(17) in line 660 is equivalent to Control-Q and is used to exit. Check with your 80-column card manual for the necessary command to exit the 80-column format.

```
45 PRINT CHR$(4);"PR#3"
70 FOR K = 1 TO 80: E$ = E$ + CHR$(32): NEXT K
80 FOR K = 1 TO 80: H$ = H$ + "-": NEXT K
90 FOR J = 1 TO 5
120 LT = 14: IF K = 5 THEN LT = 18
150 HOME: HTAB 32: PRINT "WORDTALK"
190 VTAB 1+D*3: HTAB 2
220 VTAB VP*3+1: HTAB 2
260 VTAB VP*3+1: HTAB 2
290 VTAB VP*3+1: HTAB 2
320 VTAB VP*3+1: HTAB 2
340 VTAB VP*3+1: HTAB 2
380 VTAB VP*3+1: POKE 36,HP*14-13
420 VTAB VP*3+2: POKE 36,HP*14-13: PRINT
```

```
430 VTAB VP*3+1: POKE 36,HP*14-13
660 PRINT CHR$(17)
665 VTAB 23
666 END
```

Echo Applications

Public Domain Software

By using several BASIC statements, an Echo speech synthesizer can be added to many public domain software programs. To use an Echo with public domain programs, first add the Echo software to the disk containing the programs being used. The following are the two required DOS 3.3 CATALOG entries:

```
*B 006 TEXTALKER
*B 050 TT.OBJ
```

The two Echo binary programs can also be saved on a second disk by first loading each program, determining the starting memory address and program length, and then saving each program. Although there are several ways to copy files, the technique described below can be useful in order to copy a binary program on a second disk without booting up a copy of system master disk.

(Insert disk containing TEXTALKER program)

```
BLOAD TEXTALKER
PRINT PEEK(43634) + PEEK(43635) * 256     (starting address)
37632
PRINT PEEK(43616) + PEEK(43617) * 256     (file length)
528
```

(Insert the application disk which will be using the Echo and save program)

```
BSAVE TEXTALKER, A37632, L528
```

(Insert disk containing TT.OBJ program)

```
BLOAD TT.OBJ
PRINT PEEK(43634) + PEEK(43635) * 256     (starting address)
8192
PRINT PEEK(43616) + PEEK(43617) * 256     (file length)
11956
```

(Insert the application disk which will be using the Echo and save program)

```
BSAVE TT.OBJ, A8192, L11956
```

To demonstrate how to use the Echo with other software programs, load the TEST program from disk. The easiest way to use the Echo is to load the TEXTALKER software into memory by running the program entitled TEXTALKER. The TEXTALKER program, in turn, runs the TEXTALKER.OBJECT or TT.OBJ program.

```
BRUN TEXTALKER
```

Press the RETURN key and the Echo should respond with "READY." Now enter RUN. For each number displayed on screen, the Echo pronounces the number.

The TEXTALKER can be loaded directly from the program by adding line 25:

```
25 PRINT CHR$(4);"BRUN TEXTALKER"
```

Echo Turnkey Systems

A turnkey system is one in which all that is needed is to "turn the key" and the program is up and running. A turnkey system can be created by using one of the disk utility programs previously discussed or by saving the program in question as the HELLO or greeting program. To create a turnkey system with the TALKBOARD program, first boot the system. Now load the TALKBOARD program into memory. Next, insert a blank disk and initialize the disk with the TALKBOARD program as the HELLO or greeting program:

```
LOAD TALKBOARD
```

(insert a blank disk)

```
INIT HELLO
```

Because TALKBOARD was in memory when the disk was initialized, this program now runs when the disk is booted. A second method for installing a program as the greeting program is to load boot a formatted disk, load the program, and then save the designated program using the greeting program name (which is usually HELLO):

```
LOAD TALKBOARD
SAVE HELLO
```

Of course, when using the second method, the TALKBOARD program will always run as soon as the disk is booted, even though there might be other programs of interest on the disk.

Single Switch Morse Code

The ability to use Morse Code requires not only the ability to conceptualize the relationship between a series of dots, dashes and letters, as well as a degree of reading proficiency, but also the physical ability to use a switch to enter the necessary dots and dashes. The following program is provided to illustrate one of many techniques that can be used to enter keyboard characters via single switch Morse Code.

When the program is up and running, experiment with different scan speeds (begin with a setting of 7). The scan speed for this program refers to the amount of time that is allowed in order to specify either a dot or dash. Holding the switch in a closed position for a short period of time results in a dot, while holding the key for a longer period results in a dash.

The following is a list of the codes used in the program:

A = .-
B = -...
C = -.-
D = -..
E = .

F = ..-.
G = --.
H = ...
I = ..
J = .---
K = -.-
L = .-..
M = --
N = -.
O = ---
P = .--.
Q = --.-
R = .-.
S = ...
T = -
U = ..-
V = ...-
W = .--
X = -..-
Y = -.--
Z = --..

In addition to the above .-.-.- signals the end of a message. A space is inserted between words by engaging the switch until the space character appears.

```
10 REM MORSE CODE
20 REM
30 DIM L$(26)
40 FOR K = 1 TO 26
50 READ L$(K): NEXT K
60 DATA .-,-...,-.-,-..,.
70 DATA ..-.,--.,....,..,.
80 DATA -.-,.-..,--.-.,---
90 DATA .--.,--.-,.-.,...,-
100 DATA ..-,...-,.--,-..-,-.--,--..
110 EM$ = ".-.-.-"
120 EX$ = "...-.-"
130 HOME
140 VTAB 2: HTAB 15
150 PRINT "MORSE CODE"
160 VTAB 7
170 INPUT "SCAN SPEED (1=FAST, 9=SLOW): ";ST$
180 ST = VAL(ST$): IF ST < 1 THEN ST = 4
190 VTAB 2: CALL -958
200 CC = .3*ST*20
210 VTAB 8: HTAB 6: PRINT ">"
220 IF PEEK(-16287) > 127 THEN 220
230 FOR D = 1 TO ST*50
240 IF PEEK(-16287) > 127 THEN 270
```

```
250 NEXT D
260 GOTO 360
270 FOR D = 1 TO ST*20
280 IF PEEK(-16287) < 128 THEN 320
290 NEXT D
300 M$ = M$ + CHR$(255)
310 GOTO 420
320 IF D > CC THEN L$ = L$+"-": GOTO 340
330 L$ = L$+"."
340 VTAB 8: HTAB 10: PRINT L$
350 GOTO 220
360 VTAB 8: HTAB 6: PRINT " "
370 IF L$ = EX$ THEN 490
380 IF L$ = EM$ THEN 460
390 FOR K = 1 TO 26: IF L$ = L$(K) THEN 410
400 NEXT K: GOTO 430
410 M$ = M$+CHR$(64+K)
420 VTAB 12: HTAB 1: PRINT M$
430 L$ = ""
440 VTAB 8: HTAB 1: CALL -868
450 GOTO 210
460 HOME
470 GOTO 130
480 VTAB 23
490 END
```

The letters corresponding to each code pattern are determined in line 390. When the pattern contained in L$ is equal to the pattern in L$(K), the variable K signifies the position of the letter in the alphabet. Thus, if the pattern is -- (or N), variable K is set to 14 signifying the 14th letter of the alphabet. The CHR$ value of the letter N is 78 so that CHR$(64+K) generates the letter N in lines 410 and 415.

Talking Morse Code

The MORSE CODE program is easily modified for use with an Echo to pronounce each letter as well as the message generated. If the end-of-message signal is given, the Echo presents the entire message contained in M$.

```
35 PRINT CHR$(4);"BRUN TEXTALKER"
36 G$ = CHR$(5)
37 PRINT G$"O"
38 FOR L = 1 TO 1000: NEXT L
305 PRINT G$"T "M$" "G$"O"
415 PRINT G$"T "CHR$(64+K)" "G$"O"
465 PRINT G$"T "M$" "G$"O"
466 FOR L = 1 TO 1000: NEXT L
```

Echo Cause/Effect Enhancements

The Echo can be used with virtually every type of program to provide useful and motivating feedback. The SWITCH/SCREEN CONNECT program is intended to develop a better understanding of the relationship between engaging a switch and the subsequent screen event. When the below program is run, the Echo is used to prompt the user to engage the switch when the scan cursor is within the low-resolution square displayed on screen. After the switch is engaged, low-resolution graphics, the Apple built-in sound, and the Echo are all used to provide feedback.

```
10 REM SWITCH/SCREEN CONNECT
```

```
20 REM
30 N$ = "FRED"
40 PM$ = "PRESS THE SWITCH"
50 FD = 8
60 D$ = CHR$(4): G$ = CHR$(5)
70 FOR K = 1 TO FD
80 READ FD$(K): NEXT K
90 DATA GREAT,VERY GOOD,NICE,NICE GOING,NICE WORK,
   EXCELLENT,WELL-DONE,GOOD RESPONSE
100 FOR K=1 TO 21
110 READ A
120 POKE 801+K,A
130 NEXT K
140 DATA 174,32,3,173,48,192,136,208,5,206,33,3,
    240,6,202,208,245,76,34,3,96
150 GOTO 340
160 POKE 800,101
170 POKE 801,48
180 CALL 802
190 POKE 800,76
200 POKE 801,48
210 CALL 802
220 POKE 800,60
230 POKE 801,48
240 CALL 802
250 POKE 800,52
260 POKE 801,96
270 CALL 802
280 POKE 800,60
290 POKE 801,48
300 CALL 802
310 POKE 800,52
320 POKE 801,255
330 CALL 802
340 RETURN
350 PRINT CHR$(4);"BRUN TEXTALKER"
360 PRINT G$;"O"
370 HOME: VTAB 2: HTAB 10
380 PRINT "KEY/SCREEN CONNECTION"
390 VTAB 5: HTAB 1
400 INPUT "SCAN SPEED: (1=FAST TO 9=SLOW)? ";S$
410 SP = VAL(S$)
420 IF SP < 1 OR SP > 9 THEN 350
430 PRINT
440 INPUT "NUMBER OF ITEMS? ";NX$
450 N = VAL(NX$)
460 IF N < 1 THEN 410
470 FOR K = 1 TO N
480 GR: HOME
490 RC = INT(RND(1)*15+1)
500 FOR L = 1 TO 1200: NEXT L
510 R = 0: COLOR=RC
520 IF PEEK(-16287) > 127 THEN 500
530 FOR K = 20 TO 38
540 HLIN 12,26 AT K
550 NEXT K
560 FOR K = 0 TO 12
570 POKE 800,150-K*6
580 POKE 801,150
590 CALL 802
600 COLOR=15
610 HLIN K*3,K*3+2 AT 29
```

```
620 HLIN K*3,K*3+2 AT 30
630 IF K < 4 OR K > 8 THEN 630
640 PRINT G$"T "PM$" "N$" "G$"O"
650 FOR L = 1 TO SP*10
660 KY = PEEK(-16384): IF KY > 127 THEN 670
670 IF PEEK(-16287) > 127 THEN 670
680 NEXT L: GOTO 690
690 POKE -16368,0: R = 1
700 COLOR=0
710 IF K > 3 AND K < 9 THEN COLOR=RC
720 HLIN K*3,K*3+2 AT 29
730 HLIN K*3,K*3+2 AT 30
740 IF R = 0 THEN 750
750 IF K > 3 AND K < 9 THEN 760
760 GOTO 1190
770 NEXT K: GOTO 1190
780 GOSUB 150
790 R = INT(RND(1)*FD+1)
800 PRINT G$"T "FD$(R)" "N$" "G$"O"
810 FOR K = 1 TO 15
820 COLOR=0
830 HLIN 12,26 AT 39-K
840 COLOR=RC
850 HLIN 12,26 AT 20-K
860 FOR J = 12 TO 15
870 POKE 800,200-K*10-J
880 POKE 801,5
890 CALL 802
900 NEXT J
910 NEXT K
920 FOR K = 1 TO 12
930 COLOR=0
940 VLIN 5,23 AT 11+K
950 COLOR=9
960 VLIN 5,23 AT 26+K
970 POKE 800,100-K*6
980 POKE 801,2
990 CALL 802
1000 FOR L = 1 TO 100: NEXT L
1010 NEXT K
1020 GOSUB 150
1030 FOR L = 1 TO 2000: NEXT L
1040 IF KY = 155 THEN 1060
1050 NEXT X
1060 FOR L = 1 TO 3000: NEXT L
1070 IF PEEK(-16287) > 127 THEN 1050
1080 TEXT: HOME
1090 VTAB 20: HTAB 3
1100 PRINT "(Press Return for More or Q to Quit)";
1110 POKE -16368,0: GET R$
1120 IF R$ = "Q" OR R$ = "q" THEN 1120
1130 GOTO 350
1140 END
```

To enter a student's name, N$ is set to the name in line 30. The prompt to press the switch and presented via the Echo is determined by variable PM$ in line 40. After each switch response, the Echo randomly selects one of eight statements (e.g., GREAT). These statements can be changed by modifying the DATA statement entries in line 80. If the statement in line 80 is modified, be sure that the variable FD in line 50 corresponds to the number of statements contained in the DATA statement.

In line 640, the space between PM$ (the prompt defined in line 40) and the N$ (the student's name) is needed so that the Echo treats each as a separate word.

```
640 PRINT G$"T "PM$" "N$" "G$"O"
```

Additional Echo statements could be used before the cursor reaches the target area (e.g., WAIT),. or to provide additional feedback following a correct response.

Changing Echo Speech Characteristics

Several techniques have already been discussed to show how characteristics of Echo speech (e.g., volume, pitch) can be controlled by means of Echo commands. The following illustrates how the volume and pitch can be changed in SWITCH/SCREEN CONNECT program by adding lines 355 and 356. Rather than re-running the TEXTALKER program after these additions have been made, line 350 was changed so that the Echo is re-connected. However, before line 350 is modified or used, the TEXTALKER program must have been run either by the greeting program, another application, or by the SWITCH/SCREEN CONNECT program.

```
350 PRINT D$"PR#0"
355 PRINT G$"15V"    (volume: 0 to 15 where 15 is the loudest setting)
356 PRINT G$"45P"    (pitch: 1 to 63 where 63 is the highest pitch)
```

When the program is first run, the default setting is 12 for volume and 22 for pitch.

Chapter 5

Single Switch Scan Techniques

Sequential Scanning

A sequential scanning task uses a scan or cursor to signify or somehow highlight each of the alternatives listed. When the scan reaches the last alternative or item comprising the list of alternatives, the scanning process begins anew with the first item on the list. Although it is possible to restrict the number of times the list is scanned (e.g., after 10 complete scans the task is terminated or the next item is presented), sequential scanning usually continues until a switch is engaged or the Esc key is pressed.

Scan Readiness

The SCANCOLOR program displays three colors, one of which is different. The task is to select the different color when it is scanned. If the correct alternative is selected, the screen is cleared and the color is re-shown to signify a correct response. No feedback is given following an incorrect response. The program runs until the Esc key is pressed. For each item presented, the alternatives are randomly arranged. The SCANCOLOR listing is as follows:

```
10 REM SCANCOLOR
20 REM
30 HOME
40 GR: COLOR=15
50 RA = INT(RND(1)*15+1)
60 RC = INT(RND(1)*15+1)
70 IF RA = RC THEN 50
80 R = INT(RND(1)*3+1)
90 A = 3: B = 12
100 FOR K = 1 TO 3
110 COLOR=RA
120 IF K < > R THEN 140
130 COLOR=RC: AA = A: BB = B
140 FOR J = 1 TO 15
150 HLIN A,B AT 10+J
160 NEXT J
170 A = A+12: B = B+12
180 NEXT K
190 A = 3: B = 12: P = 1
200 COLOR=15
210 HLIN A,B AT 30
220 FOR D = 1 TO 100
230 IF PEEK(-16287) > 127 THEN 320
240 IF PEEK(-16384) = 155 THEN 460
250 NEXT D
260 COLOR=0
270 HLIN A,B AT 30
280 A = A+12: B = B+12
290 IF A > 27 THEN A = 3: B = 12
300 P = P+1: IF P > 3 THEN P = 1
310 GOTO 200
320 IF R = P THEN 340
330 GOTO 440
```

```
340 COLOR=0
350 HLIN A,B AT 30
360 FOR K = 1 TO 2
370 FOR J = 1 TO 15
380 IF K = 1 THEN 410
390 HLIN AA,BB AT 10+J
400 FOR L = 1 TO 200: NEXT L: GOTO 420
410 HLIN 3,37 AT 10+J
420 NEXT J
430 COLOR=RC: NEXT K
440 FOR L = 1 TO 2000: NEXT L
450 GOTO 40
460 TEXT: HOME
470 POKE -16368,0
480 END
```

Each item can be displayed within a screen boarder by adding the following statements:

```
41 HLIN 0,39 AT 0
42 HLIN 0,39 AT 35
43 VLIN 0,35 AT 0
44 VLIN 0,35 AT 39
```

A very simple way to add sound to each cursor scan is to use CHR$(7). Each time this statement (which is the ASCII code for the bell) is encountered, a beep is sounded.

```
205 PRINT CHR$(7)
```

A similar method for highlighting each cursor scan is accomplished by the following loop:

```
325 FOR L = 1 TO 7
326 PRINT CHR$(7)
328 NEXT L
```

Touchwindow Input

The PDL(0) and PDL(1) instructions are used to sense Touchwindow input. To segment the screen into three sections which correspond to the SCANCOLOR display, line 230 is modified to read horizontal screen input. When the Touchwindow is touched as indicated by a PD value greater than 10, control is sent to line 315 and the screen color corresponding to the window position touched is set. If the value in PD is 11 to 75, P is set to 1; if the value is 76 to 150, P is set to 2; and if the value in PD is greater than 150, P is set to 3.

```
230 PD = PDL(0): IF PD > 10 THEN 315
315 P = 1: IF PD > 75 THEN P = 2
316 IF PD > 150 THEN P = 3
```

Because the Touchwindow precludes the need for sequential scanning, this part of the program can be by-passed by inserting several GOTO statements.

```
195 GOTO 230
235 GOTO 230
```

If the program is set to read Touchwindow input, and a Touchwindow is not connected, the program will act as if a response has been made. The reason for this is that when nothing is connected to the nine-pin game port, the value returned by PDL(0) is 255. If nothing is connected to the game port and the program interprets a value in PDL(0) of 10 or greater as a response, the program will logically assume that a response has been made. So as to avoid possible problems, be sure that the BASIC code is consistent with the adaptive hardware being used.

If an application is used with several devices on a frequent basis, it might be useful to add an adaptive input sensing component so that at the very beginning of a program the value returned by the nine-pin game port is read. If the value is 255, IP is set to 1 and the normal switch sensors are used; that is, PEEK(-16287) and/or PEEK(-16286); if the value in PDL(0) is less than 10, IP is set to 2 and the program then reads Touchwindow input; if PD is between 50 and 200 (and the trim settings are not out of whack), the program reads joystick input.

```
15 PD = PDL(0)
16 IF PD = 255 THEN IP = 1
17 IF PD < 10 THEN IP = 2
18 IF PD > 49 AND PD < 201 THEN IP = 3
```

The variable IP is then used throughout the program to determine what type of input is being used.

Scan Sound Techniques

To add sound to the scan component, the machine language sound subroutine discussed in the last chapter is used by adding lines 21, 22 and 23.

```
21 FOR K = 1 TO 21: READ A
22 POKE 801+K,A: NEXT K
23 DATA 174,32,3,173,48,192,136,208,5,206,
   33,3,240,6,202,208,245,76,34,3,96
```

Sound is then added to each cursor display by

```
215 POKE 800,100
216 POKE 801,200
217 CALL 802
```

To lower the pitch of the sound, modify line 215 (e.g., POKE 800,150). To increase the duration of the sound, modify line 216. The values for pitch and duration must be in the 1 to 255 range.

```
215 POKE 800,150
216 POKE 801,255
```

Chapter 4 provides a detailed description concerning how to modify sound pitch and duration.

Varying Scan Pitch

For students with visual impairments, varying the pitch of the cursor can provide useful feedback. This is accomplished in the above program by varying the pitch value in line 215 in conjunction with variable **P** which is used to designate the cursor position:

```
215 POKE 800,40+30*P
```

The above modification provides a scan pitch that decreases as the scan moves from left to right. To provide an increasing scan pitch, make the following change:

```
215 POKE 800,190-30*P
```

With the sound routine installed, auditory feedback can be provided following a correct response by adding a note/sound generating sequence beginning in line 351:

```
351 FOR L = 1 TO 15
352 POKE 800,100-L*3
353 POKE 801,50
354 CALL 802: NEXT L
```

Screen Color

What if a color monitor is not being used? Without a color monitor, several of the color shadings are not distinguishable in monochrome. For use with a monochrome monitor, set the correct answer to white (color code 15) and the incorrect answers to purple (color code 3):

```
45 RC = 15
46 RA = 3
```

Controlled and Automatic Scanning

The SCANCOLOR program uses what might be called automatic scanning in that each item in the list (colors in this situation) is automatically scanned for a specified period of time. A controlled scanning procedure can be implemented that requires the student to use the switch to move the scan. If the scan is directed to the target or correct alternative, and no further switch response is made for a specified period of time, this is treated as a correct response. Obviously, to be effective, the single switch user must understand that not activating the switch for the specified time period results in an alternative being selected.

To modify the SCANCOLOR program so that the switch is used to move the cursor and a correct response is made if the cursor scan is under the correct color and no response is made for approximately six seconds, make the following changes:

```
215 IF PEEK(-16287) > 127 THEN 215
220 FOR D = 1 TO 200
230 IF PEEK(-16287) > 127 THEN 260
255 IF R = P THEN 340
256 GOTO 200
320
330
```

The number of iterations used in the loop to evaluate switch responses is increased to 200 in line 220, and line 230 directs the program following a switch response to the routine that controls the scan. In line 255, following the completion of a loop after no switch response is made, the screen is evaluated to determine whether the scan is under the correct alternative. If this is the case, correct response feedback is given; if the scan is under an incorrect alternative and no response is made, the scan remains in that position until the switch is engaged to move the scan. In other words, the scan must be directed to the correct alternative before feedback is given.

When using single switch software, a variety of techniques must be used to determine what best meets a specific individual's needs. There are many applications where controlled or direct scanning have been extremely effective. The only cautionary note is that if the correct alternative is the first item being scanned, and the scan is not moved by means of the switch, the correct response routine is initiated if the switch is not engaged within the time period specified by the loop. For some students, this can lead to a bit of confusion.

Instead of requiring the student to move the scan to the correct alternative before feedback is given, the time delay can be used to select an alternative following either a correct or incorrect response by making these modifications:

```
255 GOTO 320
320 IF R = P THEN 340
330 GOTO 440
```

The only difficulty with the above modification is that the switch must be engaged before the loop checking for a switch response has elapsed otherwise the alternative currently being scanned is treated as a response.

Scan Latency

A student's switch behavior must be observed very closely, especially when first using scanning routines. If the student is able to engage the switch but the response is too slow, the scan interval must be increased. On the other hand, if the student is able to respond within a second or two after the correct alternative is scanned, a scan interval of 7 or 8 seconds is obviously too long.

Based on the student's observed behavior, an appropriate length of scan interval time must be determined (or at least estimated based on available observational information). Although most switch program do not include this type of feedback, it is a fairly easy task to add a scan latency tracking routine (as was done with the NILT'S NOSE program in Chapter 3) to most single switch programs. The following statements can be added to the SCANCOLOR program so that the percent of the scan interval that had elapsed for each item before a switch response is tallied and shown following the last item. An asterisk (*) is used to signify correct responses when the latency data is displayed on screen.

```
25 DIM LT$(50)
220 FOR D = 1 TO 250
441 N = N+1
442 D = INT(D / 250 * 100+.5)
443 LT$(N) = STR$(D)
444 IF R = P THEN LT$(N) = LT$(N) + "*"
480 VTAB 7: PRINT "SCAN LATENCY": PRINT
490 FOR K = 1 TO N
500 PRINT K" "LT$(K)
510 NEXT K
520 END
```

When the above routine is added, the following illustrates the type of scan latency data displayed:

```
SCAN LATENCY

1 67
2 15 *
```

```
3 54 *
```

The string array LT$(50), which is dimensioned in line 25 to hold up to 50 items, is used to store the latency results for each item. The variable N in line 441 tracks each item number and line 442 changes the scan interval value when the switch is engaged to a percentage. This value is converted to a string in line 443 and an asterisk is added to the value in LT$ if the alternative selected is correct. After the program is exited by means of the escape key, lines 480 through 510 display the latency results.

The length of the scan interval can be varied by means of a variable by specifying variable SP in line 35 at the beginning of the program, and then using this value when needed in the program:

```
35 SP = 250
220 FOR K = 1 TO SP
442 D = INT(D / 250 * 100+.5)
```

Chance Responses

Because a student is able to engage a switch during a scan task does not mean that the student actually comprehends the task or the task elements. Always remember that scan routines are essentially multiple-choice items. For a given task, the score that would be expected by chance alone is determined by

NUMBER OF ITEMS / NUMBER OF ALTERNATIVES

If 15 items are presented, and each item contains three alternatives, the score expected based on chance alone would be 15/3 or 5. If a student receives a score or 5 or 6, the possibility exists that the student's responses are similar to what would be expected by chance alone.

Although it is probably not worth the effort to create a statistical decision making model to determine if a student is responding on a random basis, the possibility of chance or random responses must always be considered when evaluating multiple-choice tasks, and especially when evaluating the single switch scan responses of a student.

Directional Scanning

The scan concept can be used in many ways to provide computer access to many tasks. The SCAN DRAW program provides a low-resolution drawing task in which a low-resolution dot is displayed in the direction of the vertical (up or down) or horizontal scan (left or right). If the scan is displayed on the left-side of the screen and the switch is engaged, a dot is displayed each time the switch is engaged toward the scan. Likewise, if the scan is at the bottom of the screen, a dot is displayed toward the scan.

In high resolution graphics, a switch graphics program could be constructed in which an arrow (which serves the same function as the "turtle" in LOGO) could be used to point to left, right, up or down. In this situation, a dot is drawn in the direction of the arrow. Another variation is to display symbols or the letters L, R, U and D at the bottom of the screen, and then to display a dot in accordance with the direction of the letter being scanned.

Although SCAN DRAW is conceptually very similar to traditional scanning in that a list of alternatives is scanned (i.e., left, right, up and down), the visual display not only shows the elements comprising the list but also graphically shows the direction. This concept could be used in the development of games (e.g., tic-tack-toe) and for tasks such as mazes in which the task is to move in different screen directions.

In terms of difficulty, if a student is not able to comprehend the task of simple scanning (e.g., the SCANCOLOR program), the student will have added difficulty conceptualizing a directional scanning procedure.

```
10 REM SCAN DRAW
20 REM
30 SP = 100
40 H = 20: V = 20
50 HOME
60 GR
70 COLOR=15: PLOT H,V
80 FOR L = 1 TO 2
90 COLOR=0
100 SX = SC-1: IF SX < 1 THEN SX = 4
110 IF L = 2 THEN SX = SC: COLOR=2
120 IF SX = 1 THEN VLIN 5,34 AT 0
130 IF SX = 2 THEN HLIN 5,34 AT 0
140 IF SX = 3 THEN VLIN 5,34 AT 39
150 IF SX = 4 THEN HLIN 5,34 AT 39
160 NEXT L
170 IF PEEK(-16287) > 127 THEN 240
180 IF PEEK(-16384) = 155 THEN 360
190 NC = NC+1: IF NC > SP THEN 210
200 GOTO 170
210 NC = O
220 SC = SC+1: IF SC > 4 THEN SC = 1
230 GOTO 80
240 IF SC = 1 THEN H = H-1
250 IF SC = 2 THEN V = V-1
260 IF SC = 3 THEN H = H+1
270 IF SC = 4 THEN V = V+1
280 IF H < 1 THEN H = 1
290 IF H > 38 THEN H = 38
300 IF V < 1 THEN V = 1
310 IF V > 38 THEN V = 38
320 COLOR=15
330 PLOT H,V
340 FOR D = 1 TO SP: NEXT D
350 GOTO 170
360 POKE -16368,0
370 TEXT: HOME
380 END
```

The variable SP in line 30 determines the scan speed. The color of the scan (either a vertical or horizontal line) is set to 2 in line 110. A counter is used to determine the length of the scan interval (variable NC in line 190). If the switch is engaged, control is sent to line 240 where SC indicates the position of the scan: if SC = 1, the scan is to the left; if 2, the scan is at the top of the screen; if 3, the scan is to the right; and if 4, the scan is at the bottom of the screen.

Depending on the position of the scan, the variables used to control the vertical (V) and horizontal (H) positions are changed in accordance with the direction of the scan. Lines 280 to 310 insure that a dot is displayed within allowable limits.

Low-resolution Scan Applications

Low-resolution Matching

There are many different tasks which can be created using low-resolution graphics. The following is a very simple matching task in which the switch is activated when the two low-resolution images are the same. If the switch is engaged, and the images are the same, a beep is sounded (lines 200 and 210). If the switch is activated and the images are not the same, nothing happens. This type of routine should be used with a liberal dose of verbal prompts to encourage activating the switch when the images are the same and to "wait" when the images are not the same.

```
10 REM LOWRES MATCH
20 REM
30 HOME
40 GR
50 FOR X = 1 TO 10
60 COLOR=15
70 S1 = INT(RND(1)*3+1)
80 S2 = INT(RND(1)*3+1)
90 H = 16: V = 6
100 ON S1 GOSUB 270,300,350
110 H = 16: V = 24
120 ON S2 GOSUB 270,300,350
130 FOR D = 1 TO 250
140 IF PEEK(-16287) > 127 THEN 190
150 NEXT D
160 GOTO 210
170 IF S1 < > S2 THEN 200
180 FOR L = 1 TO 7
190 PRINT CHR$(7): NEXT L
200 FOR L = 1 TO 2000: NEXT L
210 CALL -1994
220 NEXT X
230 TEXT: HOME
240 END
250 HLIN H,H+6 AT V+4
260 VLIN V,V+8 AT H+3
270 RETURN
280 HLIN H,H+6 AT V
290 HLIN H,H+6 AT V+8
300 VLIN V,V+8 AT H
310 VLIN V,V+8 AT H+6
320 RETURN
330 FOR K = 0 TO 3
340 PLOT H+3-K,V+K
350 PLOT H+3+K,V+K
360 NEXT K
370 HLIN H,H+6 AT V+4
380 RETURN
```

The LOWRES MATCH program first generates two random numbers between 1 and 3 in lines 70 and 80. These random values, stored in variables S1 and S2, are then used to generate one of the three low-resolution shapes stored in subroutines beginning in lines 270, 300 and 350. As an example, if S1 is set to 3, line 100 goes to the third line number in the GOSUB list which is the subroutine beginning in line 350. The statement CALL-1994 in line 230 clears the low-resolution screen after each item presentation.

To increase the probability that the two shapes are the same, add line 85:

```
85 IF RND(1) > .6 THEN S1 = S2
```

To prevent continuous input, insert a switch check after line 235.

```
235 IF PEEK(-16287) > 127 THEN 235
```

To preset the number of task items, add lines 35 and 50:

```
35 INPUT "TASK ITEMS? ";N
50 FOR X = 1 TO N
```

The following additions change the images displayed to the 15 low-resolution colors following each correct response:

```
211 FOR K = 1 TO 15
212 COLOR=K
213 H = 16: V = 6
214 ON S1 GOSUB 270,300,350
215 H = 16: V = 24
216 ON S2 GOSUB 270,300,350
217 FOR L = 1 TO 50: NEXT L
218 NEXT K
```

Following an incorrect response (e.g., the switch is engaged when the images are dissimilar), there is a slight delay before the next pair is presented. This can be changed by redirecting program control following an incorrect response from line 220 (which contains a time delay loop) to line 230 (which is used to clear the low-resolution screen):

```
170 IF S1 < > S2 THEN 210
```

In order to provide a visual "break" between task items, a second delay loop can be added after line 230.

```
236 FOR L = 1 TO 750: NEXT L
```

Although additional low-resolution shapes could be added to the program, this would decrease the frequency of two identical images appearing. This short program can best be used by modifying one of the existing low-resolution images. The following extends the figure in the third subroutine to that of a diamond:

```
371 PLOT H+K,V+3+K
372 PLOT H+6-K,V+3+K
```

The low-resolution images comprising the LOWRES MATCH program can be changed to numbers, letters, or whatever images are desired. The LOWRES NUMBER MATCH program uses graphic numbers as item alternatives:

```
10 REM LOWRES NUMBER MATCH
20 REM
30 HOME
40 GR
50 FOR X = 1 TO 10
60 IF PEEK(-16287) > 127 THEN 60
70 COLOR=15
80 S1 = INT(RND(1)*5+1)
90 S2 = INT(RND(1)*5+1)
100 IF RND(1) > .6 THEN S1 = S2
110 H = 16: V = 6
120 ON S1 GOSUB 290,330,380,440,480
130 H = 16: V = 24
```

```
140 ON S2 GOSUB 290,330,380,440,480
150 FOR D = 1 TO 250
160 IF PEEK(-16287) > 127 THEN 210
170 NEXT D
180 GOTO 250
190 COLOR=15
200 GOTO 210
210 IF S1 < > S2 THEN 240
220 FOR L =1 TO 7
230 PRINT CHR$(7): NEXT L
240 FOR L = 1 TO 2000: NEXT L
250 CALL -1994
260 NEXT X
270 TEXT: HOME
280 END
290 PLOT H+1,V
300 VLIN V,V+6 AT H+2
310 HLIN H+1,H+3 AT V+7
320 RETURN
330 HLIN H+1,H+3 AT V
340 PLOT H,V+1: PLOT H+4,V+1: PLOT H+4,V+2
350 PLOT H+3,V+3: PLOT H+2,V+4: PLOT H+1,V+5
360 PLOT H,V+6: HLIN H,H+4 AT V+7
370 RETURN
380 HLIN H,H+4 AT V
390 HLIN H,H+4 AT V+7
400 VLIN V,V+7 AT H+4
410 HLIN H+2,H+4 AT V+3
420 HLIN H+2,H+4 AT V+4
430 RETURN
440 VLIN V,V+7 AT H+3
450 PLOT H+2,V+1: PLOT H+1,V+2: PLOT H,V+3
460 HLIN H,H+4 AT V+4
470 RETURN
480 HLIN H,H+4 AT V
490 VLIN V,V+2 AT H
500 HLIN H,H+3 AT V+3
510 VLIN V+4,V+6 AT H+4
520 HLIN H,H+4 AT V+7
530 RETURN
```

The program can be modified to present horizontal and vertical lines of various sizes by deleting lines 290 to 530, and then adding lines 290 to 490:

```
DEL 290,530

290 HLIN H-1,H+1 AT V
300 RETURN
330 VLIN V-1,V+1 AT H
340 RETURN
380 HLIN H-3,H+3 AT V
390 RETURN
440 VLIN V-3,V+3 AT H
450 RETURN
480 HLIN H-5,H+5 AT V
490 RETURN
```

Yet another modification that can be made is to change the auditory feedback following a correct match by the following subroutines:

```
25 FOR K = 1 TO 21: READ A
26 POKE 801+K,A: NEXT K
```

```
27 DATA 174,32,3,173,48,192,136,208,5,206,
   33,3,240,6,202,208,245,76,34,3,96
235 GOSUB 5000
5000 POKE 800,100
5010 POKE 801,200
5020 CALL 802
5030 POKE 800,75
5040 POKE 801,150
5050 CALL 802
5060 RETURN
```

As discussed in Chapter 4, the notes generated by the subroutine beginning in line 5000 can be modified in accordance with your patience for experimentation and musical ability.

Low-resolution Discrimination

The LOWRES SCAN program provides an easy-to-modify program for presenting a three alternative low-resolution scan task. For each item presented, three low-resolution images are displayed from left to right. For every item, two of the images are the same and one is different. A horizontal line is used to scan each of the images displayed. If the switch is engaged while the dissimilar alternative is being scanned, correct feedback is given.

The program is comprised of seven different low-resolution images. These images are easily changed and/or modified and provide an excellent opportunity to create a variety of low-resolution tasks. Items are presented in a continuous loop until the program is exited by pressing the Esc key.

```
10 REM LOWRES SCAN
20 REM
30 HOME
40 VTAB 2: HTAB 15
50 PRINT "LOWRES SCAN"
60 VTAB 7
70 INPUT "SCAN SPEED (1=FAST, 9=SLOW): ";ST$
80 ST = VAL(ST$): IF ST < 1 THEN ST = 4
90 HOME
100 C = 3
110 P = 1
120 GR
130 COLOR=15
140 FOR L = 1 TO 500: NEXT L
150 VLIN 1,39 AT 13
160 VLIN 1,39 AT 27
170 RC = INT(RND(1)*7+1)
180 RI = INT(RND(1)*7+1)
190 IF RI = RC THEN 180
200 RP = INT(RND(1)*3+1)
210 FOR PL = 1 TO 3
220 GS = RI: IF RP = PL THEN GS = RC
230 IF PL = 2 THEN C = 17
240 IF PL = 3 THEN C = 31
250 ON GS GOSUB 540,560,590,610,630,660,690
260 NEXT PL
270 COLOR=15
280 HLIN (P-1)*14+4, (P-1)*14+8 AT 30
290 IF PEEK(-16287) > 127 THEN 290
300 FOR D = 1 TO ST*10
310 IF PEEK(-16287) > 127 THEN 350
320 IF PEEK(-16384) = 155 THEN 510
```

```
330 NEXT D: GOSUB 470
340 GOTO 270
350 IF RP = P THEN 390
360 FOR L = 1 TO 1000: NEXT L
370 GOTO 100
380 GOSUB 470
390 FOR K = 1 TO 7
400 PRINT CHR$(7)
410 COLOR=K*2
420 HLIN (P-1)*14+4, (P-1)*14+8 AT 30
430 NEXT K
440 FOR L = 1 TO 1500: NEXT L
450 GOSUB 470
460 GOTO 100
470 COLOR=0
480 HLIN (P-1)*14+4, (P-1)*14+8 AT 30
490 P = P+1: IF P > 3 THEN P = 1
500 RETURN
510 POKE -16368,0
520 TEXT: HOME
530 END
540 PLOT C+3,20
550 RETURN
560 PLOT C,15
570 PLOT C+6,15
580 RETURN
590 HLIN C,C+6 AT 20
600 RETURN
610 VLIN 5,25 AT C+3
620 RETURN
630 PLOT C+3,15
640 VLIN 10,20 AT C+5
650 RETURN
660 HLIN C,C+6 AT 15
670 VLIN 15,20 AT C+3
680 RETURN
690 HLIN C,C+7 AT 15
700 VLIN 15,20 AT C
710 RETURN
```

The low-resolution images are contained in the seven subroutines beginning in line 540 (see line 250 in the program which is used to call the various subroutines). The variable C indicates the beginning column for each alternative. Thus, for alternative #1 the beginning column is 3, for alternative #2 the beginning column is 17, and for alternative #3 the beginning column is 31.

The first subroutine places a low-resolution dot in column C+3 and at row 20. Remember that each PLOT sets the coordinates at PLOT COLUMN,ROW. To plot a dot at column C+3 and at row 25, modify line 540 as follows:

```
540 PLOT C+3,25
```

The subroutine in line 590 draws a horizontal line from column C to column C+6 at row 20. The subroutine in line 610 draws a vertical line from row 5 to 25 at column C+3. The length and location of each of these lines is easily modified by manipulating the HLIN and VLIN variables:

```
590 HLIN C,C+8 AT 5
610 VLIN 1,5 AT C
```

Additional statements can be added to enhance the low-resolution images:

```
615 HLIN C,C+8 AT 6
616 VLIN 7,11 AT C+8
```

The subroutine in 610 can be enhanced from a simple vertical line by the following modifications:

```
610 VLIN 11,15 AT C
615 HLIN C,C+8 AT 16
616 VLIN 17,21 AT C+8
```

As with all low-resolution graphics, changing the color is easily accomplished by setting the COLOR instruction immediately before each low-resolution image is displayed, Remember, however, to reset the color otherwise all subsequent graphics is displayed in the last color selected.

```
665 COLOR=11
675 COLOR=15
```

The above changes the color of the vertical line. To change the color of both horizontal and vertical lines, place the color statement in the first line of the routine:

```
660 COLOR=11
665 HLIN C,C+6 AT 15
```

By saving the program on disk using different file names, low-resolution scan tasks can be created using dot patterns, lines or line combinations, and a variety of low-resolution shapes.

Scan Techniques

Stimulus Scanning

Instead of using a cursor to scan alternatives, the actual target alternative can be used as the scan. When the program below is run, two low-resolution figures are displayed on the screen. Immediately below these images, the target figure is used to scan each of the alternatives. When the scan stimulus is immediately below the matching target stimulus and the switch is engaged, auditory feedback is given signifying a correct response.

If a child is having difficulty conceptualizing a three alternative scanning task, stimulus scanning might provide a useful readiness task for more abstract scanning procedures. This program presents a true/false task in which the probability of selecting a correct answer for a given item is .5. For a task comprised of 10 items, a chance score is 5.

```
10 REM STIMULUS SCANNING
20 REM
30 HOME: GR
40 FOR X = 1 TO 10
50 COLOR=15
60 S1 = INT(RND(1)*5+1)
70 S2 = INT(RND(1)*5+1)
80 IF S1 = S2 THEN 60
90 H = 8: V = 8
100 ON S1 GOSUB 350,380,430,490,530
110 H = 24
120 ON S2 GOSUB 350,380,430,490,530
130 C = S1: SP = 1: CP = 1
140 IF RND(1) > .5 THEN C = S2: SP = 2
```

```
150 H = 8: V = 30
160 ON C GOSUB 350,380,430,490,530
170 FOR D = 1 TO 250
180 IF PEEK(-16287) > 127 THEN 260
190 NEXT D
200 COLOR=0
210 ON C GOSUB 350,380,430,490,530
220 H = H+15: IF H > 25 THEN H = 8
230 CP = CP+1: IF CP > 2 THEN CP = 1
240 COLOR=15
250 GOTO 160
260 IF SP = CP THEN 280
270 GOTO 300
280 FOR L = 1 TO 7
290 PRINT CHR$(7): NEXT L
300 FOR L = 1 TO 2000: NEXT L
310 CALL -1994
320 NEXT X
330 TEXT: HOME
340 END
350 HLIN H,H+6 AT V+4
360 VLIN V,V+8 AT H+3
370 RETURN
380 HLIN H,H+6 AT V
390 HLIN H,H+6 AT V+8
400 VLIN V,V+8 AT H
410 VLIN V,V+8 AT H+6
420 RETURN
430 FOR K = 0 TO 5
440 PLOT H+3-K,V+K
450 PLOT H+3+K,V+K
460 NEXT K
470 HLIN H-3.H+9 AT V+6
480 RETURN
490 VLIN V.V+8 AT H
500 VLIN V,V+8 AT H+6
510 HLIN H,H+6 AT V+4
520 RETURN
530 FOR K = 1 TO 8
540 PLOT H+K,V+K
550 PLOT H+7-K,V+K
560 NEXT K
570 RETURN
```

The program is set to present 10 items. This number can be increased or decreased by changing the last value in the loop in line 40. To partition the screen into two sections, add the following:

```
55 VLIN 0,39 AT 19
```

Open Alternative Scanning

Most single switch scanning applications involve self-contained programs in that the program provides the alternatives, the scanning, and response feedback. There are occasions, however, when it is useful to combine a computer generated scanning routine with non-computer generated alternatives. For example, different photographs (e.g., the student's house and school building) could be positioned either below, above or on the screen (using a plastic overlay) and scanned.

Needless to say, this type of activity must be accompanied by a wealth of verbal cues such as "which is your house" or "which is your school." After the switch is engaged, the scanning is interrupted, and verbal feedback is given signifying a correct or incorrect response.

The OPENSCAN program listed below can be used to generate from 2 to 4 alternatives. In addition, the location of the scan can be set to one of three screen positions. The OPENSCAN program can be used with photographs, drawings, transparencies, concrete objects or anything else that might be appropriate. Most important, the OPENSCAN program can be used to present visual concepts and images that are most meaningful to the student.

```
10 REM OPENSCAN
20 REM
30 FOR K + 1 TO 15: E$ = E$ 4+CHR$(32): NEXT K
40 HOME: VTAB 7
50 FOR K = 1 TO 21: READ A
60 POKE 801+K,A: NEXT K
70 DATA 174.32,3,173,48,192,136,208,5,206,
   33,3,240,6,202,208,245,76,34,3,96
80 HOME: HTAB 15
90 PRINT "OPENSCAN"
100 VTAB 8
110 INPUT "ALTERNATIVES (2-4): ";AL$
120 AL = VAL(AL$): IF AL < 2 OR AL > 4 THEN 80
130 A = 2: B = 17: CN = 21: IF AL = 3 THEN A = 3:
    B = 8: CN = 14
140 IF AL = 4 THEN A = 3: B = 6: CN = 10
150 PRINT
160 INPUT "SCAN POSITION (1, 2 OR 3): ";SP$
170 SP = VAL(SP$): IF SP < 1 THEN SP = 1
180 HT = 20: IF SP = 2 THEN HT = 10
190 IF SP = 3 THEN HT = 1
200 PRINT
210 INPUT "SCAN SPEED (1=FAST TO 9=SLOW): ";ST$
220 ST = VAL(ST$): IF ST < 1 THEN ST = 1
230 N$ = LEFT$(N$,15)
240 VP = 21: IF SP = 1 THEN VP = 2
250 HOME
260 FOR L = 1 TO 1000: NEXT L
270 FOR K = HT TO HT+3
280 VTAB K: HTAB 1: CALL -868: NEXT K
290 FOR K = HT TO HT+3
300 VTAB K: HTAB A+CN*IN
310 INVERSE: PRINT LEFT$(E$,B);
320 NEXT K: NORMAL
330 POKE 800,150: POKE 801 100: CALL 802
340 POKE -16368,0
350 FOR D = 1 TO ST*20
360 IF PEEK(-16287) > 127 THEN 410
370 KY = PEEK(-16384): IF KY > 127 THEN 410
380 NEXT D
390 IN = IN+1: IF IN > AL-1 THEN IN = 0
400 GOTO 270
410 POKE -16368,0
420 IF KY = 155 THEN 550
430 VTAB VP
440 FOR L = 1 TO 1000: NEXT L
450 IN = 0
460 VTAB VP+2: HTAB 5
470 PRINT "(PRESS KEY OR SWITCH TO CONTINUE)"
480 POKE -16368,0
490 IF PEEK(-16287) > 127 THEN 520
```

```
500 KY = PEEK(-16384): IF KY > 127 THEN 520
510 GOTO 490
520 POKE -16368,0
530 IF KY = 155 THEN 550
540 GOTO 250
550 VTAB 23
560 END
```

Left-Right Open Scan

A variation of the above program is to provide a very large scan on either the left- or the right-hand side of the screen. As with the OPENSCAN program, this application can be used with photos, overlays, drawings or even objects. For example, a picture of a horse and a duck are placed before the screen. The student is asked, "Which is the duck?". If the student selects the scan highlighting the duck, the student is give immediate verbal feedback.

```
10 REM OPENSCAN-2
20 REM
30 FOR K = 1 TO 15: E$ = E$ + CHR$(32): NEXT K
40 HOME: VTAB 7
50 FOR K = 1 TO 21: READ A
60 POKE 801+K,A: NEXT K
70 DATA 174,32,3,173,48,192,136,208,5,206,
   33,3,240,6,202,208,245,76,34,3,96
80 HOME: HTAB 15
90 PRINT "OPENSCAN-2"
100 VTAB 7
110 INPUT "SCAN SPEED (1=FAST TO 9=SLOW): ";ST$
120 ST = VAL(ST$): IF ST < 1 THEN ST = 1
130 V = 1: H = 3
140 HOME
150 FOR L = 1 TO 2000: NEXT L
160 FOR K = 3 TO 20
170 VTAB V+K: HTAB H
180 INVERSE: PRINT E$
190 NEXT K
200 NORMAL
210 POKE 800,150: POKE 801,100: CALL 802
220 POKE -16368,0
230 FOR D = 1 TO ST*20
240 IF PEEK(-16287) > 127 THEN 300
250 KY = PEEK(-16384): IF KY > 127 THEN 300
260 NEXT D
270 HOME
280 H = H+20: IF H > 23 THEN H = 3
290 GOTO 160
300 POKE -16368,0
310 IF KY = 155 THEN 420
320 FOR L = 1 TO 2000: NEXT L
330 VTAB 23: HTAB 5
340 PRINT "(PRESS KEY OR SWITCH TO CONTINUE)"
350 POKE -16368,0
360 IF PEEK(-16287) > 127 THEN 390
370 KY = PEEK(-16384): IF KY > 127 THEN 390
380 GOTO 360
390 POKE -16368,0
400 IF KY = 155 THEN 420
410 GOTO 130
420 VTAB 23
430 END
```

Quadrant Scanning

Another variation of the open scanning technique is quadrant scanning. For this type of scan procedure, the screen is divided into four sections and each section is scanned in sequence. As shown in line 280 below, the scan moves from left-to-right, then from the first row to the second row. After the fourth entry in the quadrant has been scanned, the scan is positioned to the first quadrant and the scanning sequence begins anew.

```
          |
     1    |    2
          |
 ---------|----------
          |
     3    |    4
          |
```

The following line modifications to the above program partition the screen into four quadrants, and then scan each quadrant in the order specified.

```
10 REM OPENSCAN-4
90 PRINT "OPENSCAN-4"
160 FOR K = 0 TO 8
280 H = H+20: IF H > 23 THEN H = 3: V = V+12:
    IF V > 13 THEN V = 1
```

To change the scan sequence as shown below the order of the scan sequence, change line 280:

```
          |
     1    |    3
          |
 ---------|----------
          |
     2    |    4
          |
```

```
280 V = V+12: IF V > 13 THEN V = 1: H = H+20:
    IF H > 23 THEN H = 3
```

Chapter 6

High-resolution Graphics

Easy-to-Use High-resolution Programs

High-resolution Screen Displays

High-resolution graphics has played an extremely important role in the development of single switch software. From a programming standpoint, graphics does present several problems in that the programming can be more difficult and the amount of memory required can be considerable. The HIRES COLOR program listed below is relatively short, but it does illustrate how to generate a high-resolution image.

HGR	(set high-resolution graphics mode)
HCOLOR	(set high-resolution color)
HPLOT	(plot a high-resolution dot or line)

The high-resolution screen can display up to 53,760 dots using a screen comprised of up to 280 columns and 192 rows. Each time the switch is engaged, either a vertical horizontal or vertical high-resolution line is displayed. Actually, the line is comprised of five high-resolution lines as specified by the loop beginning in line 140.

```
10 REM HIRES COLOR
20 REM
30 HOME
40 HGR
50 HC = HC+1: IF HC > 7 THEN HC = 1
60 IF PEEK(-16287) > 127 THEN 60
70 IF PEEK(-16287) > 127 THEN 100
80 IF PEEK(-16384) = 155 THEN 210
90 GOTO 70
100 HCOLOR=HC
110 R = INT(RND(1)*155)
120 C = INT(RND(1)*275)
130 RN =  RND(1)
140 FOR L = 1 TO 5
150 IF RN > .5 THEN 180
160 HPLOT C+L,0 TO C+L,155
170 GOTO 190
180 HPLOT 0,R+L TO 275,R+L
190 NEXT L
200 GOTO 50
210 POKE -16368,0
220 TEXT: HOME
230 END
```

The HIRES COLOR program can be used to check (or set) a color monitor by displaying the HCOLOR code each time the switch is engaged:

```
195 VTAB 22: PRINT "COLOR CODE = "HC
```

The following high-resolution color codes are available:

CODE	COLOR
0	black

1	**green**
2	**violet**
3	**white**
4	**black**
5	**orange**
6	**blue**
7	**white**

Essentially, there are only six colors because of the eight codes (0 to 7), 0 and 4 are used for black and 3 and 7 for white. Try experimenting with different colors by changing the HCOLOR statement in line 100. To display lines using a single color such as green, change line 100:

```
100 HCOLOR=1
```

The above is one program in which a continuous switch response might be allowed. This can be accomplished by deleting line 60.

Instead of displaying a wide high resolution line, change line 140 to display a narrower high-resolution line:

```
140 FOR L = 1 TO 1
```

or try

```
140 FOR L = 1 TO 2
```

The HPLOT command can be used to specify a single dot by indicating the column, followed by the row value. The following displays a dot in column 279 and row 0 (the upper right-hand corner of the screen):

```
196 HCOLOR=3
197 HPLOT 279,0
```

Needless to say, the density of the dots can be used to produce much clearer images as compared to low-resolution graphics. However, high-resolution graphics can substantially increase the amount of memory being used, and thereby limit the size of a BASIC program that can be used.

Dot-to-Dot

The DOT-TO-DOT program is an easy-to-use high-resolution graphics program that illustrates how a single switch can be used to draw high-resolution shapes, figures and images. The program begins by reading the screen dots that are to be connected. The coordinates for each dot are read from DATA statements beginning in line 280.

The first DATA statement in line 280 signifies that a dot is to be displayed at screen position 120,20, where 120 indicates the column or horizontal screen location, and 20 signifies the row or vertical screen position. Each high resolution dot is specified by column and row screen coordinates:

```
HPLOT COLUMN,ROW
```

or

```
HPLOT 120,20           (column 120 and row 20)
```

After the screen coordinates have been read, the dots are displayed (lines 130-150). After the dots are displayed, and each time the switch is engaged (line 180), line 230 is used to draw a line from each pair of coordinates. Thus, the first pair of coordinates are 120 and 20, and the second pair of coordinates are 90, 140. Therefore, the first time the switch is engaged, a line is drawn from 120,20 to 90,140 or:

```
HPLOT 120,20 TO 90,140
```

Because the coordinates are lagged each time a line is drawn, the variable LN is used to specify the beginning and ending point of each line. When the program is first run, M(1,1)=120, M(1,2)=20, M(2,1)=90 and M(2,2)=140. For example, when LN is 1, a line is drawn from 120,20 to 90,140:

```
230 HPLOT M(LN,1), M(LN,2) TO M(LN+1,1),M(LN+1,2)
```

After all the lines have been drawn, the screen is cleared and the dots redisplayed. To modify the program, change the DATA statements to draw a different shape or pattern.

```
10 REM DOT-TO-DOT
20 REM
30 DIM M(100,2)
40 ONERR GOTO 100
50 N = 1
60 FOR K = 1 TO 2
70 READ M(N,K): NEXT K
80 N = N+1
90 GOTO 60
100 POKE 216,0: N = N-1
110 HOME
120 HGR
130 FOR K = 1 TO N
140 HPLOT M(K,1), M(K,2)
150 NEXT
160 HCOLOR=7
170 IF PEEK(-16287) > 127 THEN 170
180 IF PEEK(-16287) > 127 THEN 210
190 IF PEEK(-16384) = 155 THEN 250
200 GOTO 180
210 LN = LN+1
220 IF LN > N-1 THEN LN = 0: GOTO 120
230 HPLOT M(LN,1),M(LN,2) TO M(LN+1,1),M(LN+1,2)
240 GOTO 170
250 POKE -16368,0
260 TEXT: HOME
270 END
280 DATA 120,20
290 DATA 90,140
300 DATA 200,70
310 DATA 60,70
320 DATA 180,40
330 DATA 120,20
```

To create a square around the star add lines 331 through 335:

```
331 DATA 200,20
332 DATA 200,140
333 DATA 60,140
334 DATA 60,20
335 DATA 120,20
```

Or to draw a square delete lines 280 through 330 and modify line 335:

```
DEL 280,330
335 DATA 200,20
```

Creating High-resolution Graphics

The following program illustrates how high-resolution graphics can be used to draw geometric shapes:

```
10 REM HIRES SHAPES
20 REM
30 HOME
40 Y = 60: X = 100
50 HGR
60 HCOLOR = 7
70 GS = GS+1: IF GS > 4 THEN GS = 1
80 IF PEEK(-16287) > 127 THEN 80
90 IF PEEK(-16287) > 127 THEN 120
100 IF PEEK(-16384) = 155 THEN 150
110 GOTO 90
120 CALL 62450
130 ON GS GOSUB 180,210,240,270
140 GOTO 60
150 POKE -16368,0
160 TEXT: HOME
170 END
180 FOR K = 1 TO 60
190 HPLOT X,Y+K TO X+60,Y+K
200 NEXT K: RETURN
210 FOR K = 1 TO 30
220 HPLOT X+15,Y+15+K TO X+45,Y+15+K
230 NEXT K: RETURN
240 FOR K = 1 TO 60
250 HPLOT X+20,Y+K TO X+40,Y+K
260 NEXT K: RETURN
270 FOR K = 0 TO 23
280 HPLOT X+30-K,70+K TO X+30+K,70+K
290 NEXT K: RETURN
```

To change the color of the shapes, reset HCOLOR in line 60:

```
60 HCOLOR=2
```

Each time the switch is engaged, line 130 is used to call a subroutine to draw a geometric shape. For example, when GS is 1, control is sent to the subroutine beginning in line 180; when GS is 2, the subroutine beginning in line 210 is called; when GS is 3, the subroutine in line 240 is called; and when GS is 4, the subroutine beginning in line 270 is called.

The number of shapes can be increased by changing lines 70 and 130, and then adding several additional subroutines:

```
70 GS = GS+1: IF GS > 7 THEN GS = 1
130 ON GS GOSUB 180,210,240,270,300,330,360
300 FOR K = 1 TO 60
310 HPLOT X,Y+K TO X+K,Y+K
320 NEXT K: RETURN
330 FOR K = 43 TO 0 STEP - 1
340 HPLOT X+30-K,93-K TO X+30+K,93-K
350 NEXT K: RETURN
360 FOR K = 0 TO 43
370 HPLOT X+30-K,50+K TO X+30+K,50+K
380 NEXT K: RETURN
```

High-resolution Discrimination

The DISCRIM program is a modification of the HIRES SHAPES application task and involves the display of three shapes on screen at the same time for each item. Two of the shapes are the same and one is different. The program randomly selects the correct shape, the position of the correct shape and the incorrect alternatives. Following a correct response, the color of the scan cursor is changed. Items are presented until the Esc key is pressed.

```
10 REM DISCRIM
20 REM
30 HOME
40 Y = 40: X = 18
50 HGR
60 HCOLOR=7
70 RC = INT (RND(1)*7+1)
80 RI = INT (RND(1)*7+1)
90 IF RI = RC THEN 80
100 RP = INT ( RND(1)*3+1)
110 FOR PL = 1 TO 3
120 GS = RI: IF RP = PL THEN GS = RC
130 IF PL = 2 THEN X = 108
140 IF PL = 3 THEN X = 198
150 ON GS GOSUB 470,500,530,560,590,620,650
160 NEXT PL
170 SP = 40: P = 1
180 HCOLOR=7
190 FOR L = 0 TO 9
200 HPLOT SP,115+L TO SP+16, 115+L: NEXT L
210 FOR D = 1 TO 100
220 IF PEEK (-16287) > 127 THEN 260
230 IF PEEK (-16384) = 155 THEN 440
240 NEXT D: GOSUB 380
250 GOTO 180
260 IF RP = P THEN 290
270 GOTO 40
280 GOSUB 380
290 HCOLOR=2
300 FOR L = 1 TO 8
310 SP = 40: IF RP = 2 THEN SP = 130
320 IF RP = 3 THEN SP = 220
330 FOR L = 0 TO 9
340 HPLOT SP,115+L TO SP+16,115+L: NEXT L
350 FOR L = 1 TO 1500: NEXT L
360 GOSUB 380
370 GOTO 40
380 HCOLOR=0
390 FOR L = 0 TO 9
400 HPLOT SP,115+L TO SP+16,115+L: NEXT L
410 SP = SP+90: IF SP > 230 THEN SP = 40
420 P = P+1: IF P > 3 THEN P = 1
430 RETURN
440 POKE -16368,0
450 TEXT: HOME
460 END
470 FOR K = 1 TO 60
480 HPLOT X,Y+K TO X+60,Y+K
490 NEXT K: RETURN
500 FOR K = 1 TO 30
510 HPLOT X+15,Y+15+K TO X+45,Y+15+K
520 NEXT K: RETURN
```

```
530 FOR K = 1 TO 60
540 HPLOT X+20,Y+K TO X+40,Y+K
550 NEXT K: RETURN
560 FOR K = 0 TO 23
570 HPLOT X+30-K,70+K TO X+30+K,70+K
580 NEXT K: RETURN
590 FOR K = 1 TO 60
600 HPLOT X,Y+K TO X+K,Y+K
610 NEXT K: RETURN
620 FOR K = 43 TO 0 STEP -1
630 HPLOT X+30-K,93-K TO X+30+K,93-K
640 NEXT K: RETURN
650 FOR K = 0 TO 43
660 HPLOT X+30-K,50+K TO X+30+K,50+K
670 NEXT K: RETURN
```

The subroutines for the seven possible shapes used in the program are contained in lines 470, 500, 530, 560, 590, 620 and 650.

To present the shapes in sequential order (e.g., for the first item, the first shape is treated as the correct answer; for the second item, the second shape is the correct answer, etc), add lines 65 and 66 and modify line 70. Although the shapes are presented in sequential order, the location of the correct shape displayed on screen is still randomly determined. Shape #1 might be the correct answer, but this can appear as either the first, second or third item alternative.

```
65 RX = RX+1
66 IF RX > 7 THEN RX = 1
70 RC = RX
```

Add a "click" to each scan by using CHR$(7):

```
215 PRINT CHR$(7)
```

Following a correct response, not a great deal happens. Program feedback can be added by having the correct shape displayed (if selected) in the seven high-resolution colors on a sequential basis following a correct response:

```
145 IF RP = PL THEN XC = X: YC = Y

290 X = XC: Y = YC
300 FOR C = 1 TO 7
310 HCOLOR=C
320 ON RC GOSUB 470,500,530,560,590,620,650
310 NEXT C
320
```

Memory Move

As previously discussed, there are occasions when it is necessary to change the starting memory location of a BASIC program. If a program is fairly large (approximately 25 sectors or greater) and uses high resolution graphics, use the brief program described in the last section of Chapter 2 or the MEMORY MOVE program described in Chapter 3 to reset the beginning memory location of BASIC to either 16384 (if only page 1 of high-resolution graphics is used) or to 24576 (if both page 1 and 2 of high-resolution graphics are used).

For example, if a large program uses high-resolution graphics so that the program extends from 2049 and beyond 8191 (and into high-resolution graphics memory space), that part of the program extending into graphics space will be lost. This problem is solved by running the following program so that BASIC begins immediately after high-resolution (page 1) graphics space:

```
NEW
10 START = 16384: POKE START - 1,0
20 POKE 103, START - INT(START/256) * 256
30 POKE 104, INT (START/256)
```

If both page 1 and page 2 of high-resolution graphics are being used, set START to 24576 in line 10.

After the above BASIC starting address memory move program is run, check to make sure the starting memory location for BASIC has been changed:

```
PRINT PEEK(103) + PEEK(104) * 256
16384
```

In the above situation when the application is loaded into memory, the beginning memory location starts immediately above high-resolution page 1 graphics. As a result there is no overlap between the BASIC program and graphics memory.

If there is a problem involving memory size and high-resolution graphics, first consider using the secondary page high-resolution picture buffer when extends from 16384 to 24575. This is easily accomplished by simply using HGR2 instead of HGR.

Shape Tables

In addition to using HPLOT to draw high-resolution shapes, many programs use what are called **shape tables**. Creating a shape table from scratch is no easy task, and requires considerable experience and the BASIC Programming Reference Manual (pages 92 to 100) at the ready. However, shape tables do provide an opportunity to manipulate screen shapes in terms of size and screen position.

The SHAPE TABLE program shown below is intended to demonstrate several ways in which shapes can be manipulated on screen. When the program is first run, the shape is defined and stored in memory in lines 30 to 80. Each time the switch is engaged, the shape (a simple square) is enlarged. After 30 repetitions, the screen color is changed and the task repeated.

The key to the program is line 200 in which the shape is drawn at the specified location.

```
200 DRAW 1 AT C,R
```

In line 170, SCALE determines the size of the shape (which is a value between 0 and 255).

```
10 REM SHAPE TABLE
20 REM
30 FOR K = 1 TO 13
40 READ A
50 POKE 767+K, A: NEXT K
60 DATA 1,0,4,0,36,36,45,45,54,54,63,63,0
70 POKE 232,0
80 POKE 233,3
90 HOME: HGR
100 HC = 3
```

```
110 HCOLOR=HC
120 IF PEEK(-16287) > 127 THEN 120
130 IF PEEK(-16287) > 127 THEN 160
140 IF PEEK(-16384) = 155 THEN 270
150 GOTO 130
160 S = S+1
170 SCALE=S
180 C = C+5
190 R = R+5
200 DRAW 1 AT C,R
210 IF S < 30 THEN 120
220 S = 0: C = 0: R = 0
230 HC = HC+1
240 IF HC = 4 OR HC = 7 THEN HC = HC+1
250 IF HC > 7 THEN HC = 1
260 GOTO 110
270 POKE -16368,0
280 TEXT: HOME
290 END
```

The clarity of the shape can be enhanced by printing the same shape side-by-side:

```
205 DRAW 1 AT C+1,R
```

The position of the shape can be altered by using the ROT instruction. ROT is a value between 0 and 64 so that a ROT value of 16 rotates the shape 90 degrees, a value of 32 rotates the shape 180 degrees, etc.

```
100
110
160 SCALE=12
170 HCOLOR=0
180 XDRAW 1 AT 120,80
190 R = R+8: IF R > 64 THEN R = 0
200 ROT=R
210 HCOLOR=3
220 DRAW 1 AT 120,80
230
240
250
260 GOTO 120
```

Although shape tables can be entered directly into memory as shown above, many programs first recall the shape table from disk. Instead of lines 30 through 60 above, the following can be used to load a shape table from disk:

```
30 PRINT CHR$(4);"BLOAD SHAPE TABLE"
70 POKE 232,0
80 POKE 233,3                          (starting address of shape table)
```

The two POKE statements in lines 70 and 80 indicate to the computer where the shape table begins. Whenever a shape table is used, memory locations always indicate just where in memory the shape table is located. In this case, the shape table begins at address 768. This is probably going to be as clear as mud, but the values poked in lines 70 and 80 (0 and 3) represent decimal memory location 768 or (3 X 256) + (0 X 0). The starting location of a shape tables signifies the number of shapes contained in the table.

If using shapes from an already defined shape table, determine how many shapes are in the table by first loading the table, interrupting the program, and then peeking the value in the first shape table memory location.

```
A = PEEK(43634) + PEEK(43635) * 256  (starting shape table address)
PEEK(A)                              (number of shapes)
```

The following program is a useful shape table utility that can be used to review all the shapes comprising a shape table. The program automatically sets variable **A** to the shape table starting address in line 90. Lines 110 and 120 converts this starting decimal address to fit the allowable register value size, and then stores these values in registers 232 and 233. The first location of every shape table indicates the number of shapes in the table. In the below program variable **A** is peeked and the value returned is the total number of shapes in the table which is then stored in stored in variable MX (see line 130).

```
10 REM SHAPE TABLE REVIEW
20 REM
30 HOME
40 VTAB 2: HTAB 8
50 PRINT "SHAPE TABLE REVIEW UTILITY"
60 VTAB 8: INPUT "TABLE NAME: ";N$
70 IF N$ = "" THEN 340
80 PRINT CHR$(4);"BLOAD "N$
90 A = PEEK(43634) + PEEK(43635) * 256
100 T = INT(A/256)
110 POKE 232,A-T*256
120 POKE 233,T
130 MX = PEEK(A)
140 HOME: HGR
150 VTAB 21: PRINT "TOTAL SHAPES = "MX
160 HCOLOR=3
170 NS = NS+1: IF NS> MX THEN NS = 1
180 IF PEEK(-16287) > 127 THEN 180
190 IF PEEK(-16287) > 127 THEN 250
200 KY = PEEK(-16384): IF KY = 155 THEN 340
210 POKE -16368,0
220 IF KY = 160 THEN 320
230 IF KY > 127 THEN 250
240 GOTO 190
250 HGR
260 SCALE=1
270 C = 75: R = 100
280 VTAB 23
290 PRINT "SHAPE # = "NS"  "
300 DRAW NS AT C,R
310 GOTO 170
320 TEXT
330 NS = 0: GOTO 30
340 POKE -16368,0
350 TEXT: HOME
360 VTAB 7
370 PRINT "FILE = "N$
380 PRINT "STARTING ADDRESS = "A
390 L = PEEK(43616) + PEEK(43617) * 256
400 PRINT "FILE LENGTH = "L
360 END
```

All of the routines presented in this manual use the same procedure for reading shape tables. If desired, a variation of this routine could be used by deleting line 100, and then peeking the starting location of the table directly into locations 232 and 233:

```
100
110 POKE 232,PEEK(43634)
```

```
120 POKE 233,PEEK(43635)
```

All of the font applications have included a routine which identifies the starting point, and then pokes the appropriate values in registers 232 and 233 so that any ASCII font set can be used. If the starting location of the shape table and the number of shapes within the table are known, the shape input routine can be abbreviated as follows:

```
80 PRINT CHR$(4);"BLOAD "N$
90
100
110 POKE 232,PEEK(43634)
120 POKE 233,PEEK(43635)
130 MX = PEEK(24576)
```

To use the SHAPE TABLE REVIEW program, insert the disk containing the shape table and enter the name of the shape table file following the prompt TABLE NAME and then press RETURN. High-resolution graphics mode is set and the number of shapes in the table displayed at the bottom of the screen. If there are 5 shapes in the table, each shape appears after the switch is engaged, in sequential order, before the first shape is displayed again. This program automatically determines the number of shapes in the file.

```
                    SHAPE TABLE REVIEW UTILITY

          TABLE NAME: LASCII
```

If the name of the shape table is not known, check the disk catalog for binary files (e.g., **B 005 SHAPE NAME**) which are used to store shape tables. On the program disk, the two shape table files are recorded as

B 015 LASCII
B 010 RASCII

The program disk contains a large ASCII character font set entitled LASCII which can be accessed using the SHAPE TABLE REVIEW program. The actual shapes contained within this shape table begin with shape 30. The first 29 entries are essentially empty and correspond to ASCII Control+Key keyboard combinations. Shape 30 is an underline which can be used for scanning, and shape 31 is a solid block which can be used with various single switch applications.

Other characters in the LASCII file correspond to the usual ASCII characters. Thus, the number **0** corresponds to shape 48, the letter **A** to shape 65, and the lowercase **a** to 97.

The above program displays each shape contained in the table at screen location C,R (see line 300). The shapes could be displayed so that all appear on the screen at the same time for comparison purposes by the following modifications:

```
165 V = 20
166 HGR
250 REM
270 C = C+30: IF C > 250 THEN C = 30: R = R+40
```

The increments for C and R will depend on the size of the shapes. The above settings display eight shapes across the screen using four screen rows (i.e., the R and C settings create an 8X4 high-resolution screen matrix).

Saving Shape Tables on Disk

To save a shape table on a different disk, run the SHAPE TABLE REVIEW program. After the shapes have been read and the number of shapes displayed at the bottom of the screen, exit the program by pressing the Escape key. The starting address and the length of the file is then shown:

```
FILE = LASCII
STARTING ADDRESS = 24576
FILE LENGTH = 3404
```

If the value is 24576, insert the disk on which the shape table is to be copied, and then save the table as follows:

```
BSAVE FILE NAME, A24576, L3404
```

The starting address and file length already determined are then entered after the A and L and the file can now be saved on a different disk.

The BLOAD statement can also be used to load a shape table at an address other than the address used to originally save the program. If the original address is 24576, but a starting address of 30000 is wanted, the following would be used:

```
BLOAD FILE NAME, A30000
```

The above is useful when a binary file is loaded into a portion of memory that is used by a program. If the starting address is 2049 for a binary program, and the basic program begins at 2049 (which all do unless otherwise set), the binary program loaded at 2049 will clobber the BASIC program. This can be resolved by loading the binary program at an address not used by the program (or graphics). The program can now be saved at the new address (where the length of the file is determined as previously shown):

```
BSAVE FILE NAME, A30000,L3404
```

The next section provides additional information concerning how to save binary programs using the BSAVE instruction.

Shape Table Maker

Shapes are an important part of Applesoft and single switch programming. Once a shape table has been created, the shapes are easily manipulated on the screen by means of the DRAW, XDRAW, ROT (rotation) and SCALE commands. Based on the directions provided in the Applesoft BASIC manual, making a shape table might seem to be an extremely complicated task. And if you go about constructing shapes without a little programming help, it will be!

Although creating a shape table does require a modest amount of planning (and a little trial and error programming), the program shown below provides a relatively easy method for creating up to N number of individual shape tables and how to store the tables on disk. The SHAPE TABLE MAKER program is certainly not as involved as a commercial shape table maker, but it does offer a simple and inexpensive means for creating and storing shape tables on disk.

```
10 REM SHAPE TABLE MAKER
```

```
20 REM
30 HOME
40 BA = 24576
50 READ N
60 POKE BA,N
70 POKE BA+1,0
80 SP = SP+1
90 S = N*2+C+2
100 IF S > 255 THEN 140
110 POKE BA+SP*2,S
120 POKE BA+SP*2+1,0
130 GOTO 170
140 T = INT(S/256)
150 POKE BA+SP*2+1,T
160 POKE BA+SP*2,S-T*256
170 SD = N*2+BA+1
180 READ X
190 IF X = 9 THEN 320
200 READ Y
210 IF Y = 9 THEN Z = 1: Y = 0: YT = 0: GOTO 270
220 READ YT
230 IF YT = 9 THEN Z = 1: YT = 0: GOTO 270
240 IF YT > 3 THEN YX = YT+10: YT = 0
250 VTAB 23: HTAB 15
260 PRINT "SHAPE = "SP
270 P = YT*64+Y*8+X
280 C = C+1
290 POKE SD+C,P
300 IF YX > 9 THEN X = YX-10: YX = 0: GOTO 200
310 IF Z = 0 THEN 180
320 Z = 0
330 C = C+1
340 POKE SD+C,0
350 IF SP = N THEN 370
360 GOTO 80
370 T = INT(BA/256)
380 POKE 232,BA-T*256
390 POKE 233,T
400 HGR: COLOR=7
410 SCALE=1
420 GET D$: D=VAL(D$)
430 CALL -3086
440 IF D$ = CHR$(27) THEN 490
450 DX = DX+1
460 IF DX > N THEN DX = 1
470 DRAW DX AT 80,80
480 GOTO 420
490 TEXT: HOME
500 PRINT "START = "BA
510 PRINT "LENGTH = "SD+C+1-BA
520 END
530 REM
540 REM DATA STATEMENTS
550 REM
560 DATA 3
570 DATA 4,1,4,1,4,1,4,1,4,1,4,1,4,1,9
580 DATA 6,1,6,1,6,1,6,1,6,1,6,1,6,1,9
590 DATA 5,5,5,5,5,5,5,5,5,5,5,5,9
```

To use SHAPE TABLE MAKER, first enter the number of shapes to be created in the DATA statement in line 560. In this example, three shapes are created. Next, use a DATA statement for each shape to be created. Each shape should end with a 9 to indicate the end of that particular shape.

The numeric values in the DATA statements are the instructions for drawing high-resolution dots and moving to different screen locations. The values 0 to 3 are used to move without plotting a high-resolution dot. The values 4 through seven are used to plot a high-resolution dot and move. The values 6666 in sequence plots a dot, moves one space down, and repeats this procedure four times. The values 6,6,6,6,5,5,5,5,4,4,4,4,7,7,7,7 produces a box comprised of four downward dots, four dots draw from left to right, four upward dots, and then fours dots drawn from right to left.

The following indicates the values associated with plotting or not plotting a dot, and the direction of each subsequent move:

```
          DON'T PLOT                       PLOT

             (up)                          (up)
               0                             4
    (left) 3         1 (right      (left) 7         5 (right)
               2                             6
            (down)                        (down)
```

Consider the DATA statement in line 570:

```
570 DATA 4,1,4,1,4,1,4,1,4,1,4,1,4,1,9
```

The first value is 4 and plots a high resolution dot and moves one position up. The next value is a 1 and moves one position to the right without plotting a dot. The series of 4's and 1's is used to plot a high-resolution line in an upward direction at approximately a 45 degree angle.

To see the shapes created using the DATA statements, run the program and then enter the number (1, 2 or 3 for the above program) corresponding to the shape. The sample DATA statements in the above program draw the following shapes:

#1 = /
**#2 = **
#3 = ---

When the program is run, entering a shape table number displays the corresponding shape. To see the shapes contained in the program without clearing the screen between shapes, delete line 430.

```
430
```

When plotting a shape, be sure not to use two consecutive 0's because this will be entered as a 0 which signifies the end of the shape definition. With a little planning this should not be much of a problem. Also, because of the way shape moves are stored, a 0 or upward move without a plot, is sometimes ignored. If this happens, you might need to move either left or right before moving upward.

When the SHAPE TABLE MAKER program is run, the number of each shape is shown at the bottom of the screen as it is being read by the program. The key to the program is line 270 where combinations of shape moves are transformed to decimal equivalents. If the first two moves of a shape are 4 and 1, this would be stored as 12 (or 1X8 + 4); and if three moves are 3, 3 and 3, this would be stored as 219 or (3X64 + 3X8 + 3).

When all the shapes have been loaded, high-resolution graphics is set and the shapes are displayed in sequential order following each key press (i.e., simply press the RETURN key and each shapes will appear in sequential order). To add more shapes to the above program, remember to change the value in line 560. The following illustrates how a 4th shape (the letter N) can be added to the program.

```
430 CALL-3086
560 DATA 4
600 DATA 6,6,6,6,6,6,4,4,4,4,1,6,1,6,1,6,
    1,6,4,4,4,4,4,4,4,9
```

And this shape is the letter **A**:

```
560 DATA 5
610 DATA 1,1,5,2,3,3,5,1,5,2,3
620 DATA 3,3,3,5,1,1,1,5,2,3,3,3,3,3
630 DATA 5,1,1,1,5,2,3,3,3,3,3
640 DATA 5,5,5,5,5,2,3,3,3,3,3
650 DATA 5,1,1,1,5,2,3,3,3,3,3
660 DATA 5,1,1,1,5,9
```

To experiment with he SCALE command, reset SCALE in line 410:

```
410 SCALE=5
```

Saving Completed Shape Tables

After the shapes have been entered, run the program to see if the shapes resembled the intended images. If all is well, press ESC and the starting address and length of the table appears on screen:

```
START = 24576
LENGTH = 31
```

As is the case when using the SHAPE TABLE REVIEW program, shape tables created via the SHAPE TABLE MAKER are saved by specifying the starting address and the length of the file and then saving that segment of memory containing the shape information as a binary file. As the above program is now written, the shape table begins in address 24576 but this can be changed to meet specific programming needs. Select an appropriate name for the shape table, and then enter the name, starting address and table length. If the name is SHAPE.M, the table would be saved as:

```
BSAVE SHAPE.M, A24576, L31
```

After defining the shapes, and determining that the shapes look like the shapes (or "sort of like" the shapes) wanted, save the shape on disk. The SHAPE TABLE REVIEW program can be used to access and review the shapes created.

High-resolution Fonts

A frequent request of those involved with single switch software is a desire to have characters larger than the usual ASCII character set (pronounced As-Key and is an acronym for **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange). Very often this is useful for students with visual problems and/or to increase attention. The program disk contains two large font character sets. These character sets are contained in the binary files LASCII and RASCII. This section describes several program that can access and use the large font sets.

Normal screen characters are displayed by what is equivalent to a 7 by 5 high-resolution dot matrix. In other words, screen characters are approximately 7 high-resolution dots high and 5 high-resolution dots wide. The character in LASCII are 14 dots high and 10 dots wide. To review the character set run the SHAPE TABLE REVIEW program and then enter LASCII after the prompt.

Large Font Applications

DISPLAY FONT: This program illustrates how the LASCII character set is accessed and individual font characters displayed. When the program is run, a character font is displayed. When the switch is engaged, feedback is given by a series of beeps (line 190) and the character is displayed using the seven high-resolution color codes. The LASCII characters are displayed at screen location 125 (column or horizontal position) and 50 (row or vertical position) by means of the DRAW statements in lines 120 and 200.

```
10 REM DISPLAY FONT
20 REM
30 HOME
40 PRINT CHR$(4);"BLOAD LASCII"
50 A = PEEK(43634) + PEEK(43635) * 256
60 T = INT (A/256)
70 POKE 232,A-T*256
80 POKE 233,T
90 C = INT(RND(1)*26+65)
100 HGR: HCOLOR=7
110 SCALE=1
120 DRAW C AT 125,50
130 IF PEEK(-16287) > 127 THEN 130
140 IF PEEK(-16287) > 127 THEN 170
150 IF PEEK(-16384) = 155 THEN 240
160 GOTO 140
170 FOR CL = 1 TO 7
180 HCOLOR=CL
190 PRINT CHR$(7)
200 DRAW C AT 125,50
210 FOR D = 1 TO 200: NEXT D
220 NEXT CL
230 GOTO 90
240 POKE -16368,0
250 TEXT: HOME
260 END
```

As shown above, the large font character set is loaded into memory in line 40. The variable **A** indicates the starting address where the LASCII binary file is loaded (A = 24576). The number of shapes in the file is determined by PRINT PEEK(A). The variable C is a random number ranging from 65 to 90 which corresponds to the shapes **A** to **Z**. To use the DISPLAY FONT program to present the characters 0 to 9, line 90 is set to

```
90 C = INT(RND(1)*10+48)
```

Lower-case letters are displayed by setting C as follows:

```
90 C = INT(RND(1)*26+97)
```

Following a switch response, additional feedback could be given by displaying the target font character at various screen locations by incorporating the XDRAW statement:

```
170 FOR K = 1 TO 25
175 H = INT(RND(1)*200+25)
176 V = INT(RND(1)*100+25)
180
200 DRAW C AT H,V
215 XDRAW C AT H,V
220 NEXT K
```

Additional modifications include changing the HCOLOR setting in line 100, and experimenting with the ROT instruction which rotates shapes increments from 0 (no rotation) to 64 (380 degrees rotation):

```
170 FOR K = 1 TO 4
175
176
195 HCOLOR=0
200 DRAW C AT 125,50
214 ROT=16*K
215 XDRAW C AT 125,50
220 NEXT K
```

The shape values in the LASCII file range from 30 to 126. Shapes 1 to 29 correspond to Control+Key combinations and are essentially "empty" shapes. The complete list of LASCII shapes is as follows:

SHAPE #	CHARACTER
30	_
31	■
32	SPACEBAR
33	!
34	"
35	#
36	$
37	%
38	&
39	'
40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6

55	7
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^
95	_
96	`
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j

107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	|
125	}
126	~

SWITCH FONT: Before using the SWITCH FONT program, set N$ to the student's name in line 70. The SWITCH FONT program first reads a series of words from DATA statements beginning in line 590. After the N words have been read, the Echo TEXTALKER program and the LASCII character set are loaded into memory. The program is currently set to present the words sequentially in line 200, where RW is set to X, and X specifies each of the items from 1 to N.

The words are shown using the LASCII character set by means of the subroutine beginning in line 310. To use this subroutine the column or horizontal position for each letter is set by HX, and the row or vertical position of each word displayed is indicated by V (V is set to 75 in line 50). In line 370 the starting horizontal position is adjusted for the length of the string being presented so that the string contained in W$ is centered.

The word is displayed on the screen until the switch is engaged. When the switch is closed, the message "Very good" followed by the students name is presented via the Echo. This is followed by the message "The word is", and then the word displayed. The word is then shown on screen using the various high-resolution colors.

After response feedback has been given and the screen is cleared, there is a delay of approximately 3 seconds. If the switch is engaged while in this loop, the loop is re-initiated. This was installed to emphasize the need to wait until a screen word appeared before engaging the switch. The length of the delay loop is controlled by variable ST in line 60.

Although the task does not demand or measure understanding of the words presented, an attempt should be made to develop an association between a single switch response and program feedback. When working with a student, provide ample verbal cues. For example, when the screen clears say, "Now wait for the word. Watch the screen and wait for the word."

```
10 REM SWITCH FONT
20 REM
30 DIM W$(50), W(50)
40 D$ = CHR$(4): G$ = CHR$(5)
```

```
50 V = 75
60 ST = 150
70 N$ = "JANET"
80 ONERR GOTO 110
90 N = N+1
100 READ W$(N): GOTO 90
110 POKE 216,0: HOME
120 N = N-1
130 PRINT D$; "BRUN TEXTALKER"
140 PRINT D$;"BLOAD LASCII"
150 A = PEEK(43634) + PEEK(43635) * 256
160 T = INT (A/256)
170 POKE 232,A-T*256
180 POKE 233,T
190 FOR X = 1 TO N
200 RW = X: GOTO 240
210 RW = INT(RND(1) * N + 1)
220 IF W(RW) = 1 THEN 210
230 W(RW) = 1
240 HGR: HCOLOR=7
250 SCALE=1
260 W$ = W$(RW)
270 HX = 130-10*LEN(W$)/2
280 H = HX
290 GOSUB 310
300 GOTO 380
310 FOR L = 1 TO LEN(W$)
320 C$ = MID$(W$,L,1)
330 C = ASC(C$)
340 DRAW C AT H,V
350 H = H+13
360 NEXT L
370 RETURN
380 IF PEEK(-16287) > 127 THEN 410
390 IF PEEK(-16384) = 155 THEN 560
400 GOTO 380
410 PRINT G$"T VERY GOOD "N$" "G$"O"
420 FOR D = 1 TO 500: NEXT D
430 PRINT G$"T THE WORD IS "W$" "G$"O"
440 FOR CL = 1 TO 7
450 H = HX
460 HCOLOR=CL: GOSUB 310
470 FOR D = 1 TO 200: NEXT D
480 NEXT CL
490 FOR D = 1 TO 2000: NEXT D
500 HGR
510 FOR D = 1 TO ST
520 IF PEEK(-16384) = 155 THEN 560
530 IF PEEK(-16287) > 127 THEN 510
540 NEXT D
550 NEXT X
560 POKE -16368,0
570 TEXT: HOME
580 END
590 DATA SINGLE SWITCH,READING,MATH,YES,GREAT!
```

Each time a character is displayed, H is incremented in line 350 by 13 to accommodate the next character and to provide spacing between characters. The between-letter spacing can be decreased by changing this line. Because each character is 10 dots wide, the smallest increment should be 11:

```
350 H = H+11      (small character spacing)
```

```
350 H = H+15        (wide character spacing)
```

An Echo prompt can be added to the delay routine so that during the pause phase of the program, the Echo prompt WAIT FOR THE WORD is given:

```
530 IF PEEK(-16287) > 127 THEN 535
531 GOTO 540
535 PRINT G$"T WAIT FOR THE WORD "G$"O"
536 GOTO 510
```

To present the words in random order delete line 200. To use SWITCH FONT with a Touchwindow (and this is how the program was originally used), change line 380 as follows:

```
380 IF PDL(0) > 5 THEN 410
```

When using this program with a Touchwindow, be sure that the device is connected or the program will assume that a response has been made each time line 380 is executed.

FONT MATCH: The FONT MATCH program selects two characters and displays both at the same time. If the characters are the same and a switch is engaged, a series of beeps is given to indicate a correct response. In the LASCII shape table, the letter **A** corresponds to the shape value 65 and the letter **Z** to a shape value of 90.

```
10 REM FONT MATCH
20 REM
30 HOME
40 HGR
50 HCOLOR=7
60 SCALE=1
70 PRINT CHR$(4);"BLOAD LASCII"
80 A = PEEK(43634) + PEEK(43635)*256
90 T = INT(A/256)
100 POKE 232,A-T*256
110 POKE 233,T
120 FOR X = 1 TO 10
130 S1 = INT(RND(1)*26+65)
140 S2 = INT(RND(1)*26+65)
150 IF RND(1) > .5 THEN S1 = S2
160 DRAW S1 AT 125,50
170 DRAW S2 AT 125,90
180 FOR D = 1 TO 500
190 IF PEEK(-16287) > 127 THEN 220
200 NEXT D
210 GOTO 270
220 IF S1 < > S2 THEN 250
230 FOR L = 1 TO 7
240 PRINT CHR$(7): NEXT L
250 FOR L = 1 TO 2000: NEXT L
260 IF PEEK(-16287) > 127 THEN 260
270 HGR
280 FOR L = 1 TO 1000: NEXT L
290 NEXT X
300 TEXT: HOME
310 END
```

A routine can be added that results in the re-presentation of items following an incorrect response:

```
155 HGR
156 FOR D = 1 TO 500: NEXT D
205 IF S1 = S2 THEN 155
220 IF S1 < > S2 THEN 155
```

Lower-case letters can be used by changing lines 130 and 140 as follows:

```
130 S1 = INT(RND(1)*26+97)
140 S2 = INT(RND(1)*26+97)
```

And the numbers 0 through 9 can be used by the following changes:

```
130 S1 = INT(RND(1)*10+48)
140 S2 = INT(RND(1)*10+48)
```

The program is easily transformed into a nonverbal task by specifying non-letter characters in the LASCII file. The following statements generate the values 4, 5, 6 and 7 which correspond to the keyboard characters #, $, % and &.

```
130 S1 = INT(RND(1)*4+35)
140 S2 = INT(RND(1)*4+35)
```

The next set of statements generate the values 9, 10, 11, AND 12 which correspond to the keyboard characters (,) *, and +.

```
130 S1 = INT(RND(1)*4+40)
140 S2 = INT(RND(1)*4+40)
```

To modify the program so that a Touchwindow response indicates a match, line 190 is changed as follows:

```
190 IF PDL(0) > 25 THEN 220
```

FONT WRITER: The FONT WRITER program can be used to display all the large font characters. As shown in the program listing, this program begins by loading the Echo TEXTALKER program. If an Echo is not available, insert a GOTO statement in line 35 to by-pass the TEXTALKER program.

```
35 GOTO 70      (Insert if an Echo is not being used)
```

However, the FONT WRITER is best used with an Echo. The reason for this is that whenever the switch is engaged, the information shown on screen is presented via the Echo. The program could be used by entering the student's name and then having the student engage the switch to activate the Echo. Additional letters, words, phrases or sentences could then be added and used in a similar fashion.

The FONT WRITER is an extremely easy-to-use word processor which combines large screen characters with synthesized speech. One important limitation, however, is that the screen displays a limited amount of information (17 rows by 5 columns of LASCII characters). Although the program could be modified to print out text and even to file text, be aware that the high-resolution graphics does use a large chuck of memory so that possible program length is limited.

```
10 REM FONT WRITER
20 REM
30 D$ = CHR$(4): G$ = CHR$(5)
40 PRINT D$;"BRUN TEXTALKER"
50 FOR L = 1 TO 1000: NEXT L
60 PRINT G$;"O"
```

```
70 HOME
80 PRINT D$;"BLOAD LASCII"
90 A = PEEK(43634) + PEEK(43635) * 256
100 T = INT(A/256)
110 POKE 232,A-T*256
120 POKE 233,T
130 HGR
140 SCALE=1
150 H = 25: V = 25
160 HCOLOR=7: DRAW 30 AT H,V
170 IF PEEK(-16287) > 127 THEN 450
180 KY = PEEK(-16384): IF KY > 127 THEN 200
190 GOTO 170
200 POKE -16368,0
210 HCOLOR=0: DRAW 30 AT H,V
220 IF KY = 152 THEN S$ = "": GOTO 130
230 IF KY = 155 THEN 480
240 IF KY = 149 THEN 390
250 IF KY = 136 THEN 420
260 IF KY = 138 THEN V = V+25: GOTO 400
270 IF KY = 141 THEN H = 25: V = V+25: GOTO 400
280 IF KY = 139 THEN V = V-25: GOTO 430
290 DRAW 30 AT H,V
300 DRAW 31 AT H,V: DRAW 31 AT H,V+5
310 P = ((H-25) / 13 + 1) + (V - 25) / 25 * 17
320 IF P = 1 THEN L$ = "": GOTO 340
330 L$ = LEFT$(S$,P-1)
340 R$ = MID$(S$,P+1,120)
350 S$ = L$ + CHR$(KY) + R$
360 IF KY = 160 THEN 380
370 HCOLOR=7
380 DRAW KY-128 AT H,V
390 H = H+13: IF H > 240 THEN H = 25: V = V+25
400 IF V > 125 THEN H = 25: V = 25
410 GOTO 160
420 H = H-13: IF H < 25 THEN H = 25: V = V-25
430 IF V < 25 THEN V = 25: H = 25
440 GOTO 160
450 PRINT
460 PRINT G$"T "S$" "G$"O"
470 GOTO 160
480 TEXT: HOME
490 END
```

The FONT WRITER program records the screen characters as a single string by means of the string variable S$ (see line 350). After the ASCII equivalent of a keyboard key has been recorded, the corresponding letter shape is determined by subtracting 128 from variable KY (line 380). The current position of the characters within this string is determined by the variable P in line 310. When the switch is engaged, the S$ string is presented via the Echo in line 460.

The FONT WRITER is especially useful for entering short phrases and sentences, and then having the student engage the switch after the word or phrase has been entered.
Following a response, the screen is cleared and the cursor returns to the beginning position by pressing the Control+X key combination (see line 220).

FONT FEEDBACK: This application illustrates how a variety of program components can be integrated into a program that provides considerable auditory and visual feedback following each switch response. The program is comprised of a series of statements (or single words) contained in DATA statements beginning in line 930. The first statement indicates that message that is displayed via the LASCII character set. The second statement is the message generated by the Echo. Using corresponding statements allows for the specification of Echo speech (see line 930).

When entering DATA statements, the first field is the font message or word displayed by the monitor and the second field is synthesized by the Echo.

```
930 DATA "HELLO, SEAN!","HELLO, SHAWN!"

XXX DATA FONT FIELD, ECHO FIELD
```

Each DATA statement for the FONT FEEDBACK program must have two fields; the two fields must be separated by a comma; and if a comma is contained within a field, the field must be enclosed with quotation marks as shown in line 930.

Following a switch response, several notes are played, the message is synthesized and then displayed in different colors. Before the next statement is displayed, the switch device must be in an open position for approximately three seconds.

Notice that at the bottom of the screen a decimal is displayed. This corresponds to the up/down movement of the joystick. This was included to provide information as to whether the student was exerting any downward joystick movement, even though the movement might not have been sufficient to close or activate the switch. In this way the variable resistance of the joystick potentiometer provides information relating to the intensity of the switch response.

The FONT FEEDBACK program was designed to integrate several feedback systems into a single program. If used to primarily enhance reading skill, consider replacing the "charge" subroutine with a REM statement in line 730. There is a time when feedback is essential, and a time when too much feedback can be overwhelming. One great advantage of BASIC software is that the amount and type of feedback can be controlled.

The DATA statement information is presented sequentially. To present the N messages or words in random order, delete line 490.

```
10 REM FONT FEEDBACK
20 REM
30 ST = 50
40 V = 75
50 FOR K = 1 TO 21
60 READ A
70 POKE 801+K,A
80 NEXT K
90 DATA 174,32,3,173,48,192,136,208,5,206,
   33,3,240,6,202,208,245,76,34,3,96
100 GOTO 300
110 POKE 800,101
120 POKE 801,48
130 CALL 802
140 POKE 800,76
150 POKE 801,48
160 CALL 802
170 POKE 800,60
180 POKE 801,48
190 CALL 802
200 POKE 800,52
210 POKE 801,96
220 CALL 802
```

```
230 POKE 800,60
240 POKE 801,48
250 CALL 802
260 POKE 800,52
270 POKE 801,255
280 CALL 802
290 RETURN
300 DIM W$(50),W(50),E$(50)
310 D$ = CHR$(4): G$ = CHR$(5)
320 ONERR GOTO 350
330 N = N+1
340 READ W$(N), E$(N): GOTO 330
350 POKE 216,0: HOME
360 N = N-1
370 IF PEEK(999) = 99 THEN 420
380 POKE 999,99
390 PRINT D$;"BRUN TEXTALKER"
400 PRINT D$;"BLOAD LASCII"
410 GOTO 430
420 PRINT CHR$(4);"PR#0"
430 A = PEEK(43634) + PEEK(43635) * 256
440 T = INT(A/256)
450 POKE 232,A-T*256
460 POKE 233,T
470 PRINT G$;"O"
480 FOR X = 1 TO N
490 RW = X: GOTO 530
500 RW = INT(RND(1)*N+1)
510 IF W(RW) = 1 THEN 500
520 W(RW) = 1
530 HGR: HCOLOR=7
540 SCALE=1
550 W$ = W$(RW)
560 E$ = W$(RW)
570 HX = 140-LEN(W$)*13/2
580 H = HX
590 GOSUB 610
600 GOTO 680
610 FOR L = 1 TO LEN(W$)
620 C$ = MID$(W$,L,1)
630 C = ASC(C$)
640 DRAW C AT H,V
650 H = H+13
660 NEXT L
670 RETURN
680 IF PEEK(-16287) > 127 THEN 730
690 VTAB 22: PRINT PDL(1)"  "
700 IF PDL(1) > 200 THEN 730
710 IF PEEK(-16384) = 155 THEN 900
720 GOTO 680
730 GOSUB 110
740 PRINT G$"T "E$" "G$"O"
750 FOR D = 1 TO 500: NEXT D
760 FOR CL = 1 TO 7
770 H = HX
780 HCOLOR=CL: GOSUB 610
790 FOR D = 1 TO 200: NEXT D
800 NEXT CL
810 FOR D = 1 TO 2000: NEXT D
820 HGR: HOME
830 FOR D = 1 TO ST
840 IF PEEK(-16287) > 127 THEN 830
```

```
850 VTAB 22: PRINT PDL(1)"  "
860 IF PDL(1) > 200 THEN 830
870 IF PEEK(-16384) = 155 THEN 900
880 NEXT D
890 NEXT X
900 POKE -16368,0
910 TEXT: HOME
920 END
930 DATA "HELLO, SEAN!","HELLO, SHAWN!"
940 DATA HOW ARE YOU TODAY?,HOW ARE YOU TODAY?
950 DATA PULL THE LEVER.,PULL THE LEVER.
960 DATA "VERY GOOD, SEAN!","VERY GOOD, SHAWN!"
970 DATA "TRY AGAIN, SEAN.","TRY AGAIN, SHAWN."
980 DATA "GOOD WORK, SEAN!","GOOD WORK, SHAWN!"
```

The routine for presenting routines in contained in the subroutine beginning in line 610. The entire routine could be compacted so as to increase programming speed by entering the routine as a single statement:

```
610 FOR L = 1 TO LEN(W$): DRAW ASC(MID$(W$,L,1)) AT H,V:
    H = H+13: NEXT: RETURN
```

The one provision that must be taken into consideration when using a large font character set is that the amount of information that can be displayed on the screen at any given time is limited. Although a large character set is appealing, if a student can use normal ASCII characters, this should be the preferred video display mode. Chapter 7 provides additional techniques for using the LASCII character set in conjunction with scanning routines.

Combining Text and Graphics

A high-resolution font set is easily combined with graphics. The binary file containing the character set is first loaded, then the routine to display characters is added to the application. To illustrate, load the DOT-TO-DOT program from the program disk. Next, add the statements shown below. When the program is run, the message PRESS THE SWITCH is displayed at the bottom of the screen.

```
115 PRINT CHR$(4);"BLOAD LASCII"
116 A = PEEK(43634) + PEEK(43635) * 256
117 T = INT(A/256)
118 POKE 232,A-T*256
119 POKE 233,T
121 SCALE=1
122 W$ = "PRESS THE SWITCH"
123 H = 120-10*LEN(W$)/2
124 GOSUB 125: GOTO 130
125 FOR L = 1 TO LEN(W$)
126 C = ASC(MID$(W$,L,1))
127 DRAW C AT H,145
128 H = H+13
129 NEXT L:RETURN
```

To display the message at the top of the screen, change the vertical starting point in line 127:

```
127 DRAW C AT H,0
```

If regular size ASCII characters are needed for a high-resolution graphics screen, the RASCII font set can be loaded and used. Erase the program in memory using NEW and run the following:

```
10 PRINT CHR$(4);"BLOAD RASCII"
20 HGR: HCOLOR=7
30 SCALE=1
40 POKE 232,PEEK(43634): POKE 233,PEEK(43635)
50 HPLOT 50,75 TO 200,75
60 DRAW 65 AT 45,65
70 DRAW 66 AT 200,65
```

The above draws a line segment and indicates the beginning of the line by the letter "A" and the ending of the line by the letter "B". Additional text can be added by decoding strings of data as illustrated by these additions:

```
15 READS T$
16 DATA "THIS IS A LINE."
80 FOR L = 1 TO LEN(T$)
90 C = ASC(MID$(T$,L,1))
100 DRAW C AT 40+L*7,90
110 NEXT L
```

Variable Size Font Applications

The advantages of using a ready-made large font character set are threefold: 1) The characters are highly visible, have a predictable size, and are displayed on the screen fairly quickly. However, if you try to further increase the size of the font characters contained in the LASCII file by means of the SCALE instruction, you will find that the result is not particularly readable. This is because of how the LASCII characters were written. By connecting lines and writing from left-to-right and then from right-to-left, the resulting characters are not especially amenable to the SCALE instruction.

In order to use the SCALE instruction effectively, each character or shape should be written from left-to-right and with no downward connecting lines. If the SHAPE TABLE MAKER is used, the following would be entered to plot the ASCII character **N**:

```
3130 DATA 5,1,1,1,5,2,3,3,3,3,3
3140 DATA 5,1,1,1,5,2,3,3,3,3,3
3150 DATA 5,5,1,1,5,2,3,3,3,3,3
3160 DATA 5,1,5,1,5,2,3,3,3,3,3
3170 DATA 5,1,1,5,5,2,3,3,3,3,3
3180 DATA 5,1,1,1,5,2,3,3,3,3,3
3190 DATA 5,1,1,1,5,2,3,3,3,3,3
```

The above procedure has been used to produce a character set that can generate characters any size wanted (within reason, of course), from small to absolutely huge. The RASCII or "regular" ASCII character set contained on the program disk was constructed by only plotting dots from left to right, and not connecting rows of dots within each character. The result is characters that are easily enlarged using the SCALE instruction.

Run the RASCII DISPLAY FONT to see how characters are displayed on screen. To change the size of the characters reset the variable SC which is used to specify the SCALE value in line 100.

```
100 SC=5
```

SCALE can be set to a value ranging from 1 to 255, and a scale value of

0 will result in the maximum scale size. As the SCALE size increases, the number of characters that can be displayed on a given line decreases. The following provides a general guideline for the number of letters that can be displayed on a line for various SCALE values.

SCALE	**CHARACTERS PER LINE**
1	45
2	22
3	15
4	11
5	9
6	7
7	6
8	6
9	5
10	4
11	4
12	4
13	3
14	3
15	3

In terms of type size as expressed in points, normal ASCII characters have a point size of approximately 18, where 1 point is 1/72 of an inch. Thus, a point size of 18 results in uppercase letters 1/4 inch in size. Scaling characters to 2 results in a point size of 36 which is the point size of the LASCII character set. A scale value of 4 results in a point size of 72 (or characters 1 inch high); and a scale value of 8 results in a point size of 144 (or uppercase letters 2 inches high). The LASCII character set has a point size of 36 which results in uppercase letters 1/2 inch high.

The following illustrates the scaling of the letter **A** using the RASCII character set with the SCALE value is set to 3. Each time the letter is displayed, the vertical beginning point for displaying the letter is moved so that the first vertical point is row 76, the second starting point is row 77 and the third vertical starting point is row 78. As a result, after three passes, the scaled letter is filled-in and thereby displayed on screen as a solid character.

Variable Size Single Character Applications

Variable SC in line 100 is set to 8 so that the resulting size of each character displayed is 2 inches or a point size of 144. As was already said, this technique results in some distortion as the size of scale size is increased, but there is no doubt that this approach can be used to display letters that are extremely visual! For an even larger display, set SC to 15. Note, however, that as the size of the characters increase, the number of characters that can be displayed at a give time decreases.

```
10 REM RASCII DISPLAY FONT
20 REM
30 HOME
40 PRINT CHR$(4);"BLOAD RASCII"
50 A = PEEK(43634) + PEEK(43635) * 256
60 T = INT (A/256)
70 POKE 232,A-T*256
80 POKE 233,T
90 SC = 8
100 C = 64
```

```
110 C = C+1: if C > 90 THEN C = 65
120 HGR: HCOLOR=7
130 GOTO 180
140 SCALE=SC: ROT=0
150 FOR K=1 TO SC
160 DRAW C AT 125,50+K
170 NEXT: RETURN
180 IF PEEK(-16287) > 127 THEN 130
190 IF PEEK(-16287) > 127 THEN 170
200 IF PEEK(-16384) = 155 THEN 240
210 GOTO 190
220 GOSUB 140
230 FOR D = 1 TO 1500: NEXT D
240 GOTO 110
250 POKE -16368,0
260 TEXT: HOME
270 END
```

To add speech to the program, make these modifications:

```
35 PRINT CHR$(4);"BRUN TEXTALKER"
225 PRINT CHR$(C)
226 PRINT "THE LETTER IS "CHR$(C)
```

If the TEXTALKER program has already been run, line 35 can be set to reconnect the Echo:

```
35 PRINT CHR$(4);"PR#0"
```

To use lowercase letters, change lines 100 and 110 as follows:

```
100 C = 97
110 IF C > 122 THEN C = 97
```

To select and display letters randomly, change line 110:

```
110 C = INT(RND(1)*26+65)
```

To use numbers instead of letters, set variable C as follows:

```
100 C = 47
110 C = C+1: IF C > 57 THEN C = 48
```

Joystick Input

For one student who used a modified version of the above program, the most appropriate response seemed to be a pull motion using a joystick. The RASCII DISPLAY FONT program can be modified to accommodate this type of input by these changes:

```
35 PT = PDL(1)+50
125 IF PDL(1) > PT THEN 125
195 IF PT = 305 THEN 200
196 IF PDL(1) > PT THEN 220
```

The variable PT determines the initial setting of PDL(1) when in the neutral position, and then adds 50 to set the threshold value needed to initiate a switch response. If the initial value of PDL(1) is 130, then the joystick must be moved so that a value greater than 180 is reached before a switch response is recorded.

Line 125 requires that the joystick be in the neutral position before an actual response can be made. Line 195 by-passes the PDL(1) statement if the joystick is not connected as indicated by a PT value of 305. Line 196 reads the joystick response and transfers control to line 220 if PDL(1) is greater than variable PT.

Variable Size Word Display

The RASCII SWITCH FONT program displays a word (or string of characters) following each switch response. This is accomplished by the routine in lines 310 to 340. Note variable H in line 280. This sets the beginning column position for each character displayed based on the SCALE value (i.e., variable SC) and the number of characters in the string variable W$.

```
10 REM RASCII SWITCH FONT
20 REM
30 DIM W$(50), W(50)
40 D$ = CHR$(4): G$ = CHR$(5)
50 V = 75
60 ST = 150
70 N$ = "MIKE"
80 ONERR GOTO 110
90 N = N+1
100 READ W$(N): GOTO 90
110 POKE 216,0: HOME
120 N = N-1
130 PRINT D$;"BRUN TEXTALKER"
140 PRINT D$;"BLOAD LASCII"
150 A = PEEK(43634) + PEEK(43635) * 256
160 T = INT (A/256)
170 POKE 232,A-T*256
180 POKE 233,T
190 FOR X = 1 TO N
200 RW = X: GOTO 240
210 RW = INT(RND(1)*N+1)
220 IF W(RW) = 1 THEN 210
230 W(RW) = 1
240 HGR: HCOLOR=7
250 W$ = W$(RW): GOSUB 270
260 GOTO 350
270 SC = 4
280 H = 130 - SC*6*(LEN(W$)/2+1) - 6
290 IF H < -SC*6 THEN H = -SC*6
300 SCALE=SC: ROT=0
310 FOR K = 1 TO SC: FOR L = 1 TO LEN(W$)
320 C = ASC(MID$(W$,L,1))
330 DRAW C AT H+L*SC*6,V+K
340 NEXT: NEXT: RETURN
350 IF PEEK(-16287) > 127 THEN 380
360 IF PEEK(-16384) = 155 THEN 520
370 GOTO 350
380 PRINT G$"T VERY GOOD "N$" "G$"O"
390 FOR D = 1 TO 500: NEXT D
400 PRINT G$"T THE WORD IS "W$" "G$"O"
410 FOR CL = 1 TO 3
420 HCOLOR=CL: GOSUB 270
430 FOR D = 1 TO 200: NEXT D
440 NEXT CL
450 FOR D = 1 TO 2000: NEXT D
460 HGR
470 FOR D = 1 TO ST
```

```
480 IF PEEK(-16287) > 127 THEN 470
490 IF PEEK(-16384) = 155 THEN 520
500 NEXT D
510 NEXT X
520 POKE -16368,0
530 TEXT: HOME
540 END
540 DATA MRS. SMITH,CAR,SONG,SUMMER
```

To display each word following a switch response, make these changes:

```
250 W$ = W$(RW)
350 IF PEEK(-16287) > 127 THEN 350
355 IF PEEK(-16287) > 127 THEN 380
370 GOTO 355
385 GOSUB 270
470
480
490
500
```

The following program changes result in each word or phrase being displayed with SCALE set to 2, but the word is not filled in by a second pass. When the second pass is encountered, or when K in line 325 is greater than 1, control is sent to the newly created switch sensing routine in line 345. Now each time the switch is engaged, the full letter is displayed and presented via the Echo.

```
10 REM RASCII-APPLICATIONS
250 W$ = W$(RW)
260
270 SC = 2
325 IF K > 1 THEN GOSUB 345
340 NEXT: NEXT
341 GOTO 380
345 IF PEEK(-16287) > 127 THEN 345
350 IF PEEK(-16287) > 127 THEN 355
351 GOTO 360
355 PRINT G$" "CHR$(C)" "G$"O"
356 RETURN
410
420
430
440
```

Emphasized Character Display

To display characters at a normal text size but in an emphasized mode, each character is displayed twice when the SCALE is set to 1, but the second time the character is displayed the beginning position is lagged one dot by setting the loop in line 310:

```
270 SC = 1
310 FOR K = 1 TO 2: FOR L = 1 TO LEN(W$)
```

Advantages and Disadvantages of Variable Font Characters

The advantage of the variable font procedure is that there is maximum flexibility in terms of character size. On the other hand, there are several disadvantages: 1) the expanded characters must be plotted using a certain format (i.e., plotting characters from left to right and no upward or downward connected dots); 2) varying character size requires additional software modifications to handle the number of characters that can be presented on a single line; 3) the characters become somewhat distorted with increasing size so that an **M** scaled to 1 is somewhat different than an **M** scaled to 3; and 4) each character is displayed at the specified scale size a number of times that is equal to the scale size thereby reducing program speed. If the scale vale is 6, the character is actually displayed 6 times. This, as one might expect, can substantially reduce the speed of the program and make certain applications using this approach quite slow.

To display the characters a bit more rapidly, the display routine could confined to a single line:

```
490 FOR K = 1 TO SC: FOR L = 1 TO LEN(W$): DRAW
    ASC(MID$(W$,L,1)) AT H+L*SC*6,V+K: NEXT: NEXT: RETURN
```

Inverse Screen Display

All of the single switch programs considered thus far have used traditional white (or one of the available low- or high-resolution colors) characters on a black screen background. For certain students, an inverse screen image might be useful. To illustrate how this can be accomplished, first load the READ program from the program disk.

After the program has been loaded, add line 175 and then run the program.

```
175 INVERSE
```

When the program is run with the INVERSE instruction, the characters are displayed in inverse screen mode. If lowercase letters are used, and an 80-column card is installed, enter PR#3 (if the card is installed in slot #3) before running the program.

To display the items on a white screen add the following:

```
55 FOR K = 1 TO 40: WB$ = WB$+" ":NEXT
170 VL = 7
176 FOR K = 1 TO 22: VTAB K: PRINT WB$
320 NORMAL
340 PRINT CR$: INVERSE
```

Line 55 creates a string comprised of 40 spaces. Line 176 then displays the WB$ string so that the screen background is changed to white. The above simple modifications provides a definite visual alternative to the usual white on black video mode.

The above approach can also be used to display low- and high-resolution images in inverse mode. To accomplish this the screen must first be set to white, then the images displayed with either COLOR or HCOLOR set to 0. Load the RASCII DISPLAY FONT program and make these modifications:

```
85 HGR
100 HCOLOR=7
105 FOR K = 0 TO 159: HPLOT 0,K TO 279,K: NEXT
106 HCOLOR=0
```

The key to the modification is line 105 which changes the high-resolution screen from black to white. The HCOLOR is then set to 0 in line 106 so that the characters are displayed in black on a white screen background.

To reverse a low-resolution graphics screen, insert the following after the GR instruction and change the COLOR codes throughout the program so that codes of 0 are set to 15, and codes of 15 are changed to something other than 15.

```
41 COLOR=15: INVERSE
42 FOR K = 0 TO 39
43 HLIN 0,39 AT K: NEXT
```

Add the above routine to the NILT program and then change line 130 so that the COLOR is set to 7.

Cued Language

If a student is able to use the keyboard, or is just beginning to use a keyboard, a variable size font display can provide a source for many useful language activities. The Cued Language program shown below presents a series of words using the RASCII font set. Each time a word is presented, the outline of the word is shown. The task is to enter each letter of the word shown in sequence. If the first word is **FRED**, the first letter entered is **F**. The outline provides not only a cue, but the program will only provide feedback when the appropriate letter in the letter sequence is pressed. For students having difficulty differentiating between keys, or to provide students with a specific series of keystrokes, this activity provides a very clear and directed language activity. This program is also useful for presenting an initial language activity using the keyboard or a transition activity from a single switch to the keyboard.

Each time the appropriate letter is pressed, the outline of the letter is filled and the letter presented via the Echo. After the last letter has been entered, the entire word is re-displayed in the various high-resolution colors. The program is currently set to present the words in the order read by DATA statements. To present the words in random order, delete line 210.

Before running the program, set the name variable in line 60 to the appropriate name. Variable V in line 50 signifies the beginning vertical screen row for displaying words. The size of the RASCII font used is determined by variable SC in line 270. Experiment with different SC values, but be aware that as the font size increases the number of characters that can be display across the screen decreases.

```
10 REM CUED LANGUAGE
20 REM
30 DIM W$(50),W(50)
40 D$ = CHR$(4): G$ = CHR$(5)
50 V = 75
60 N$ = "FRED"
70 ONERR GOTO 100
80 N = N+1
90 READ W$(N): GOTO 80
100 POKE 216,0: HOME
110 N = N-1
120 PRINT D$;"BRUN TEXTALKER"
130 PRINT D$;"BLOAD RASCII"
140 A = PEEK(43634)+PEEK(43635)*256
150 T = INT(A/256)
160 POKE 232,A-T*256
170 POKE 233,T
180 FOR X = 1 TO N
190 HOME
200 RW = X: GOTO 240
```

```
210 RW = INT(RND(1)*N+1)
220 IF W(RW) = 1 THEN 210
230 W(RW) = 1
240 HGR: HCOLOR=7
250 W$ = W$(RW): GOSUB 270
260 GOTO 360
270 SC = 4
280 H = 130-SC*6*(LEN(W$)/2+1)-6
290 IF H < -SC*6 THEN H = -SC*6
300 SCALE=SC: ROT=0
310 FOR L = 1 TO LEN(W$)
320 C = ASC(MID$(W$,L,1))
330 DRAW C AT H+L*SC*6,V
340 NEXT L
350 RETURN
360 FOR L = 1 TO LEN(W$)
370 C$ = MID$(W$,L,1)
380 VTAB 20: HTAB 1
390 GET KY$
400 IF ASC(KY$) = 27 THEN 670
410 IF KY$ = C$
420 GOTO 390
430 PRINT G$"T "C$" "G$"O"
440 GOSUB 460: GOTO 490
450 FOR K = 1 TO SC
460 C = ASC(C$)
470 DRAW C AT H+L*SC*6,V+K
480 NEXT K: RETURN
490 NEXT L
500 PRINT G$"T VERY GOOD "N$" "G$"O"
510 FOR D = 1 TO 500: NEXT D
520 PRINT G$"T THE WORD IS "W$" "G$"O"
530 FOR CL = 1 TO 3
540 FOR L = 1 TO LEN(W$)
550 C$ = MID$(W$,L,1)
560 HCOLOR=CL: GOSUB 450
570 NEXT L
580 NEXT CL
590 FOR D = 1 TO 750: NEXT D
600 HGR
610 IF PEEK(-16287) > 127 THEN 610
620 IF PEEK(-16287) > 127 THEN 660
630 KY = PEEK(-16384): IF KY = 155 THEN 670
640 IF KY > 127 THEN 660
650 GOTO 620
660 NEXT X
670 POKE -16368,0
680 TEXT: HOME
690 END
700 DATA FRED,RADIO,CHAIR,MICKEY MOUSE
710 DATA CHURCH,HOUSE,HENRY,MAMA,GRANDMA
720 DATA PIZZA,HAMBURGER,JUICE,SNACK,CEREAL
```

The following changes converts the program to single switch use so that each time the switch is engaged, one character in the sequence is filled in:

```
390 IF PEEK(-16287) > 127 THEN 390
395 IF PEEK(-16287) > 127 THEN 430
400 IF PEEK(-16384) = 155 THEN 670
```

Echo Pronunciation

If there is a discrepancy between how a word should be pronounced and how the word is actually pronounced by the Echo, the following shows how to input a display form and an Echo form for each stimulus word used in the program. When this modification is used, each DATA statement must have a stimulus word (i.e., the word that is displayed on screen) and a corresponding Echo word (i.e., the word that is synthesized by the Echo). In the below example, N is set to 2 because this is the number of stimulus words read (i.e., SOCCER and RADIO) See Chapter 4 for more information regarding the Echo and Echo applications.

```
30 DIM W$(50),W(50),E$(50)
90 READ W$(N), E$(N): GOTO 80
245 E$ = E$(RW)
520 PRINT G$"T THE WORD IS "E$" "G$"O"
700 DATA SOCCER, SOCK ER
710 DATA RADIO, RAY DEE O
```

High-resolution Design Applications

High-resolution Geometric Graphics

Although low-resolution graphics are easier to use from a programming standpoint, high-resolution graphics allows for the creation of more elaborate screen images. The following program illustrates how to use high-resolution graphics in a single switch format to create a variety of geometric shapes and images. The HIRES GRAPHICS program contains two major subroutines: the first subroutine (line 120) is used to read switch input. The second subroutine (line 150) is used to draw a triangle. Before this subroutine is called, five variables must be set (see lines 130 and 140): **RE** (the number of lines drawn), **FD** (the length of the line), **RT** (the angle for drawing the next line), **X** (the horizontal axis position of the line), **Y** (the vertical axis position of the line).

```
10 REM HIRES GRAPHICS
20 REM
30 HOME
40 HGR
50 HCOLOR=3
60 C = .0174532
70 REM
80 REM ENTER GRAPHICS ROUTINES
90 REM
100 REM ROUTINE #1
110 REM
120 GOSUB 2000
130 RE = 3: FD = 55: RT = 120
140 X = 100: Y = 50
150 GOSUB 1000
160 GOSUB 2000
170 CALL 62450
500 GOTO 120
510 POKE -16368,0
520 TEXT: HOME
530 END
1000 FOR J = 1 TO RE
1010 FOR K = 1 TO 2
1020 IF K = 1 THEN D = FD: GOTO 1060
```

```
1030 AA = 360-RT: AG = AA+AG
1040 IF AG < 361 THEN 1090
1050 AG = AG-360: GOTO 1090
1060 T = INT(X + COS(AG*C) * D + .5)
1070 U = INT(Y - SIN(AG*C) * D + .5)
1080 HPLOT X,Y TO T,U: X = T: Y = U: AA = 0
1090 NEXT K: NEXT J
1100 RETURN
2000 IF PEEK(-16287) > 127 THEN 2000
2010 IF PEEK(-16287) > 127 THEN 2040
2020 IF PEEK(-16384) = 155 THEN 510
2030 GOTO 2010
2040 RETURN
```

To draw a square in the upper left-hand corner of the screen, add these lines:

```
170 FD = 60: RT = 90: RE = 4
171 X = 0: Y = 0
172 GOSUB 1000
173 GOSUB 2000
```

And to draw a circle, add these lines:

```
180 FD = 7: RT = 18: RE = 20
181 X = 170: Y = 100
182 GOSUB 1000
183 GOSUB 2000
```

High-resolution Files

As with low-resolution graphics, it is possible to create and save high-resolution screens. While a low-resolution screen is comprised of 1,024 bytes, a high-resolution consists of 8,192 bytes. As a result of this, storing a high-resolution screen on disks requires an ample amount of space: approximately 34 sectors. Because high-resolution files are quite large, loading a file from disk to the screen does take a bit of time. The following program can be used to create high-resolution graphics screens, and to store the high-resolution images on disk:

```
10 REM HIRES AUTHOR
20 REM
30 HGR
40 CC = 7
50 HOME: VTAB 22
60 PRINT "U=UP     D=DOWN   L=LEFT   R=RIGHT"
70 PRINT "E=ERASE  S=SAVE   G=GET    C=COLOR"
80 PRINT "Q=QUIT   W=WRITE  X=DON'T WRITE"
90 H = 139: V = 80
100 HCOLOR=CC: HPLOT H,V
110 X = PEEK(-16384)
120 IF X > 127 THEN 170
130 FOR L = 1 TO 8: NEXT L
140 IF CC = 0 THEN HCOLOR=7
150 IF CC > 0 THEN HCOLOR=0
160 HPLOT H,V: GOTO 100
170 X$ = CHR$(X-128): POKE -16368,0
180 IF X$ = "Q" THEN 490
190 IF X$ = "E" THEN 30
200 IF X$ = "C" THEN 360
210 IF X$ = "S" OR X$ = "G" THEN 410
```

```
220 IF X$ = "W" THEN D = 0: GOTO 100
230 IF X$ = "X" THEN D = 1: GOTO 100
240 IF D = 1 THEN HCOLOR = 0: HPLOT H,V
250 IF X$ = "U" THEN V = V-1
260 IF X$ = "D" THEN V = V+1
270 IF X$ = "L" THEN H = H-1
280 IF X$ = "R" THEN H = H+1
290 IF H < 0 THEN H = 0
300 IF H > 279 THEN H = 279
310 IF V < 0 THEN V = 0
320 IF V > 159 THEN V = 159
330 IF D = 1 THEN 100
340 HPLOT H,V
350 GOTO 100
360 HOME: VTAB 23: HTAB 5
370 PRINT "(MAKE SELECTION AND PRESS RETURN)"
380 VTAB 21: INPUT "COLOR ( 1 TO 7)? ";CC$
390 CC = VAL(CC$)
400 GOTO 50
410 HOME: VTAB 22
420 INPUT "FILE NAME? ";F$
430 HOME
440 IF X$ = "G" THEN 470
450 PRINT CHR$(4);"BSAVE";F$",A8192,L8192"
460 GOTO 40
470 PRINT CHR$(4);"BLOAD";F$
480 GOTO 40
490 POKE -16368,0
500 TEXT: HOME
510 END
```

To draw a black image on a white screen, add the following lines before running the program and then select the X (DON’T WRITE) option:

```
31 FOR K = 8192 TO 16383
32 POKE K,255: NEXT K
```

As shown in line 450, the primary high-resolution screen begins at address 8192 and is comprised of 8,192 bytes (A8192 indicates the starting memory address and L8192 signifies the file length).

High-resolution Page Switch

The two high-resolution screens are often used in conjunction to give the illusion of movement or simply to quickly present two distinct high-resolution screens. As is the case with low-resolution graphics, soft switches are used to switch between page 1 (memory locations 8192 to 16383) and page 2 (memory locations 16384 to 24575) of high-resolution graphics. The following program illustrates how to switch between page 1 and page 2 high-resolution screens.

```
10 REM HIRES PAGE SWITCH
20 REM
30 HGR2
40 HCOLOR=7
50 HPLOT 250,0 TO 125,150
60 HGR
70 HPLOT 0,0 TO 125,150
80 POKE -16302,0
90 IF PEEK(-16287) > 127 THEN 90
100 IF PEEK(-16384) = 155 THEN 190
110 IF PEEK(-16287) > 127 THEN 130
```

```
120 GOTO 100
130 POKE -16299,0
140 IF PEEK(-16287) > 127 THEN 140
150 IF PEEK(-16287) > 127 THEN 170
160 GOTO 150
170 POKE -16300,0
180 GOTO 90
190 POKE -16368,0
200 TEXT
210 END
```

The HIRES PAGE SWITCH program can be modified to read high-resolution files from disk, and then switch between different high-resolution screens. Because each high-resolution file requires considerable memory (at least 34 sectors), a separate disk might be dedicated to a program that uses many high-resolution files. To modify the above program to read a high-resolution file from disk, and then to switch between page 1 and page 2 high-resolution screens, make the following modifications:

```
50
70 F$ = "HI-RES FILE NAME"
75 PRINT CHR$(4);"BLOAD";F$
```

The high-resolution file in F$ is displayed on screen using page 1 of high-resolution memory. Each switch response alternates between page 1 and page 2 (which is blank unless otherwise set) screens. As noted before, high-resolution files can be identified in the catalog in that each has approximately 34 sectors of memory. Files containing shape tables, on the other hand, are often comprised of fewer sectors.

Chapter 7

Single Switch Math

Developing Math Skills

The purpose of single switch software is to gradually develop cognitive, learning and academic skills. If a student has the cognitive ability, there is absolutely no reason why regular curricular materials and content cannot be modified for use in a single switch format.

When using academic software, or when considering its use, considerable effort must be made to insure that the material is at the appropriate difficulty level. If the material is too easy or too difficult, learning will not take place. The goal should be to incorporate content that can be mastered.

If a student cannot match letters, providing more complicated reading tasks will probably have little impact on learning or achievement...other than increasing the student's frustration level. Likewise, if a student is capable of mastering higher level academic tasks, restricting this student to cause/effect single switch software, no matter how impressive the graphics, is a tremendous waste of intellectual potential.

As discussed in Chapter 5, a major concern of single switch scan programs is the very characteristic that makes this approach so beneficial: the multiple-choice format. For two alternatives, a score of 50% correct can be attributed to chance; for three alternatives, the chance score is 33%; for four alternatives, the chance score is 25%; and for five alternatives, the chance score is 20%.

Although there is no firm rule for deciding appropriate content difficulty level, the following factors should be considered:

1) The student has demonstrated an ability to master content at a lower difficulty level. If the student has not mastered simple primary addition facts such as **2 + 3**, there is little reason to present difficult multiplication facts (e.g., **8 X 6**);

2) The student's responses suggest an understanding of the content;

3) The student is able to select the correct alternative beyond a chance level of responding without verbal prompts;

Feedback

How and when to provide feedback following a single switch response must always be considered in relation to a specific student's needs. Although there are no absolutes with respect to feedback, three points should be kept in mind: First, negative feedback such as sad faces, sad tunes, fog horns etc. should be avoided. For children and students using single switch software, the goal is to reinforce correct responses and not "punish" incorrect selections. In short, emphasize the positive rather than the negative.

Second, the purpose of software is to enhance learning and not to display the wonderfulness of computer technology by a long and involved sound and graphics following each correct response. Following a correct response, something should happen on screen to indicate that the correct answer was selected. However, this "something" should be short and to the point.

Third, there are occasions when, following an incorrect response, the correct answer is displayed in order to provide corrective feedback and thereby enhance learning. This type of feedback is frequently useful with older students when teaching specific skills (e.g., primary facts, content area vocabulary).

Number Matching Readiness

The NUMBER COUNT program is a relatively simple cause/effect task involving the numbers 1 to 5. Each switch response causes a number to be displayed in low-resolution graphics on screen in sequential order. After the sequence of numbers has been presented, the sequence begins again with N set to 1.

```
10 REM NUMBER COUNT
20 REM
30 HOME
40 GR
50 V = 17: H = 17
60 N = N+1: IF N > 5 THEN N = 1
70 IF PEEK(-16287) > 127 THEN 70
80 IF PEEK(-16287) > 127 THEN 110
90 IF PEEK(-16384) = 155 THEN 150
100 GOTO 80
110 GR
120 COLOR=15
130 ON N GOSUB 180, 220,270,330,370
140 GOTO 60
150 POKE -16368,0
160 TEXT: HOME
170 END
180 PLOT H+1,V
190 VLIN V,V+6 AT H+2
200 HLIN H+1,H+3 AT V+7
210 RETURN
220 HLIN H+1,H+3 AT V
230 PLOT H,V+1: PLOT H+4,V+1: PLOT H+4,V+2
240 PLOT H+3,V+3: PLOT H+2,V+4: PLOT H+1,V+5
250 PLOT H,V+6: HLIN H,H+4 AT V+7
260 RETURN
270 HLIN H,H+4 AT V
280 HLIN H,H+4 AT V+7
290 VLIN V,V+7 AT H+4
300 HLIN H+2,H+4 AT V+3
310 HLIN H+2,H+4 AT V+4
320 RETURN
330 VLIN V,V+7 AT H+3
340 PLOT H+2,V+1: PLOT H+1,V+2: PLOT H,V+3
350 HLIN H,H+4 AT V+4
360 RETURN
370 HLIN H,H+4 AT V
380 VLIN V,V+2 AT H
390 HLIN H,H+3 AT V+3
400 VLIN V+4,V+6 AT H+4
410 HLIN H,H+4 AT V+7
```

```
420 RETURN
```

A modification of the above program is a two item number matching task. In this situation, the stimulus or target shape is displayed, followed by two alternatives, one of which is identical to the stimulus shape while the second alternative is dissimilar. Each of the alternative shapes is scanned by means of a horizontal line. If the switch is engaged while the shape identical to the stimulus shape is being scanned, correct feedback is provided:

```
10 REM NUMBER MATCH
20 REM
30 HOME
40 GR
50 FOR X = 1 TO 5
60 COLOR=15
70 SC = 8: SP = 1
80 S1 = INT(RND(1)*5+1)
90 S2 = INT(RND(1)*5+1)
100 IF S1 = S2  THEN 90
110 C = S1: CP = 1
120 IF RND(1) > .5 THEN C = S2: CP = 2
130 H = 17: V = 5
140 N = C: GOSUB 160
150 GOTO 180
160 ON N GOSUB 420,460,510,570,610
170 RETURN
180 H = 28: V = 22
190 N = S1: GOSUB 160
200 H = 26: V = 22
210 N = S2: GOSUB 160
220 HLIN SC,SC+4 AT 32
230 FOR D = 1 TO 150
240 IF PEEK(-16287) > 127 THEN 330
250 IF PEEK(-16384) = 155 THEN 390
260 NEXT D
270 COLOR=0
280 HLIN SC,SC+4 AT 32
290 SC = SC+18: SP = 2
300 IF SC > 30 THEN SC = 8: SP = 1
310 COLOR=15
320 GOTO 220
330 IF CP < > SP THEN 360
340 FOR L = 1 TO 5
350 PRINT CHR$(7): NEXT L
360 FOR L = 1 TO 1000: NEXT L
370 CALL -1994
380 NEXT X
390 POKE -16368,0
400 TEXT: HOME
410 END
420 PLOT H+1,V
430 VLIN V,V+6 AT H+2
440 HLIN H+1,H+3 AT V+7
450 RETURN
460 HLIN H+1,H+3 AT V
470 PLOT H,V+1: PLOT H+4,V+1: PLOT H+4,V+2
480 PLOT H+3,V+3: PLOT H+2,V+4: PLOT H+1,V+5
490 PLOT H,V+6: HLIN H,H+4 AT V+7
500 RETURN
510 HLIN H,H+4 AT V
520 HLIN H,H+4 AT V+7
530 VLIN V,V+7 AT H+4
```

```
540 HLIN H+2,H+4 AT V+3
550 HLIN H+2,H+4 AT V+4
560 RETURN
570 VLIN V,V+7 AT H+3
580 PLOT H+2,V+1: PLOT H+1,V+2: PLOT H,V+3
590 HLIN H,H+4 AT V+4
600 RETURN
610 HLIN H,H+4 AT V
620 VLIN V,V+2 AT H
630 HLIN H,H+3 AT V+3
640 VLIN V+4,V+6 AT H+4
650 HLIN H,H+4 AT V+7
660 RETURN
```

Reviewing Subroutines

By deleting lines 50 through 380 (or the entire FOR/NEXT loop), and then adding lines 50 to 120 and line 170, the single switch can be used to review each of the low-resolution subroutines. If more subroutines are added, N should be set to revert back to 1 (line 90) if the maximum number of subroutines in the program is exceeded. In addition, the beginning line of each subroutine must be added to the GOSUB list in line 160:

```
50 IF PEEK(-16287) > 127 THEN 50
60 IF PEEK(-16287) > 127 THEN 90
70 IF PEEK(-16384) = 155 THEN 390
80 GOTO 60
90 N = N+1: IF N > 5 THEN N = 1
100 H = 17: V = 15
110 CALL -1994
120 COLOR=15
160 ON N GOSUB 420,460,510,570,610
170 GOTO 50
```

Number Concept

A very important stage in the progression of single switch math skills is number concept. Before a student is able to understand a concept such as 3 + 2, the student must be able to conceptualize that the number three represents a group of three items. One method for assessing and teaching this important skill, and a skill that is required before beginning computational problems, is to use the NUMBER CONCEPT program.

When this program is run, a number between 1 and 5 is selected and low-resolution blocks corresponding to this number are displayed on the screen. Immediately below the low-resolution blocks are the numbers 1 to 5 which are scanned from left to right using a horizontal cursor. The task is to select the number that corresponds to the number of blocks displayed. If the correct number is selected, feedback is given. Following each correct response, the correct or target number is highlighted by changing the low-resolution screen color of this value (lines 370 to 410).

```
10 REM NUMBER CONCEPT
20 REM
30 H = 5
40 FOR X = 1 TO 5
50 R = INT (RND(1)*H+1)
60 NC = 0
70 GR: COLOR=15
80 HOME
```

```
90 FOR K = 1 TO H
100 ON K GOSUB 460,540,630,720,780
110 COLOR=15
120 NEXT K
130 FOR K = 0+(2.5-R*.5) TO (R-1)+2.5-R*.5
140 FOR J = 1 TO 5
150 HLIN K*8,K*8+6 AT 10+J
160 NEXT J
170 NEXT K
180 FOR L = 1 TO 500: NEXT L
190 AD = 0: IF NC = 0 THEN AD = AD+1
200 LC = NC*8+(5-H)*4+AD
210 HLIN LC,LC+4 AT 32
220 AD = 0
230 POKE -16368,0
240 FOR D = 1 TO 100
250 IF PEEK(-16287) > 127 THEN 330
260 KY = PEEK(-16384): IF KY > 127 THEN 330
270 NEXT D
280 COLOR=0
290 HLIN LC,LC+4 AT 32
300 NC = NC+1: IF NC = H THEN NC = 0
310 COLOR=15
320 GOTO 190
330 POKE -16368,0
340 IF KY = 155 THEN 1170
350 IF R = NC+1 THEN 370
360 GOTO 420
370 FOR KC = 1 TO 5
380 COLOR=10+NC
390 ON R GOSUB 460,540,630,720,780
400 FOR L = 1 TO 500: NEXT L
410 NEXT KC
420 NEXT X
430 TEXT: HOME
440 END
450 K = 0
460 Z = 3
470 IF H = 4 THEN Z = 7
480 IF H = 3 THEN Z = 11
490 IF H = 2 THEN Z = 15
500 VLIN 22,28 AT Z
510 PLOT Z-1,23
520 HLIN Z-1,Z+1 AT 29
530 RETURN
540 Z = 8
550 IF H = 4 THEN Z = 12
560 IF H = 3 THEN Z = 16
570 IF H = 2 THEN Z = 20
580 HLIN Z+1,Z+3 AT 22
590 PLOT Z,23: PLOT Z+4,23: PLOT Z+4,24
600 PLOT Z+3,25: PLOT Z+2,26: PLOT Z+1,27
610 PLOT Z,28: HLIN Z,Z+4 AT 29
620 RETURN
630 Z = 16
640 IF H = 4 THEN Z = 20
650 IF H = 3 THEN Z = 24
660 HLIN Z,Z+4 AT 22
670 HLIN Z,Z+4 AT 29
680 VLIN 22,29 AT Z+4
690 HLIN Z+2,Z+4 AT 25
700 HLIN Z+2,Z+4 AT 26
```

```
710 RETURN
720 Z = 24
730 IF H = 4 THEN Z = 28
740 VLIN 22,29 AT Z+3
750 PLOT Z+2,23: PLOT Z+1,24: PLOT Z,25
760 HLIN Z,Z+4 AT 26
770 RETURN
780 Z = 32
790 HLIN Z,Z+4 AT 22
800 VLIN 22,24 AT Z
810 HLIN Z,Z+3 AT 25
820 VLIN 26,28 AT Z+4
830 HLIN Z,Z+4 AT 29
840 RETURN
```

The program continues until the number of items specified in line 40 have been presented. To present 10 items instead of 5, modify line 40:

```
40 FOR X = 1 TO 10
```

or use a variable to specify the number of items:

```
35 N = 10
40 FOR X = 1 TO N
```

or

```
35 HOME: VTAB 5
36 INPUT "ITEMS? ";N
40 FOR X = 1 TO N
```

Time Duration Input

The majority of single switch programs use either simple cause/effect input (e.g., the switch causes a specific screen event to happen) or automatic scanning (e.g., a routine is used to scan possible alternatives from a list of possibilities). In addition, as was already discussed, controlled or indirect scanning is sometimes used in which an input device is used to scan a list of alternatives and a time delay used to select an alternative selection.

Another technique that can be used, and one that requires a very sophisticated response, incorporates a time duration routine to input a number of switch responses within a specified interval. The number of times the switch has been engaged while in the time duration interval is then used as the actual response.

As an example, a four second interval is created in which the student can press the switch from 1 to 5 times. If the switch is pressed once, the number 1 is displayed; if the switch is pressed twice, the number 2 is displayed, etc.

The key to the INTERVAL INPUT program is the switch routine contained in lines 90 to 170. This is a timing loop that counts the number of times the switch is engaged while in the loop. If the switch is engaged three times, N is incremented to 3. After the loop has been completed, if N is greater than 0, the low-resolution graphics number corresponding to N is displayed via the GOSUB routine in line 220.

Each time the loop is encountered, a READY prompt appears at the bottom of the screen. Each time the switch is engaged while in the loop, and asterisk is displayed. If the switch is not engaged while in the loop, the screen is cleared and after a delay the READY prompt is shown again. If, following the READY prompt, the switch is engaged three times, an asterisk appears following each switch response:

READY: * * *

After the interval has been completed, the three asterisks signify that the number 3 has been selected and this number is then displayed on screen in low-resolution graphics.
The Esc key is used to exit the program.

```
10 REM INTERVAL INPUT
20 REM
30 HOME
40 GR
50 FOR L = 1 TO 2000: NEXT L
60 HOME
70 VTAB 21
80 PRINT "READY: ";
90 FOR D = 1 TO 200
100 IF PEEK(-16287) > 127 THEN 130
110 IF PEEK(-16384) = 155 THEN 250
120 GOTO 170
130 N = N+1
140 IF N > 5 THEN N = 5: GOTO 190
150 PRINT "* ";
160 IF PEEK(-16287) > 127 THEN 160
170 NEXT D
180 IF N = 0 THEN 30
190 GR
200 HOME
210 COLOR=15
220 ON N GOSUB 280,330,390,460,510
230 N = 0
240 GOTO 50
250 TEXT: HOME
260 POKE -16368,0
270 END
280 H = 20: V = 10
290 VLIN V,V+6 AT H
300 PLOT H-1,11
310 HLIN H-1,H+1 AT V+7
320 RETURN
330 H = 18: V = 10
340 HLIN H+1,H+3 AT V+2
350 PLOT H,H+3: PLOT H+4,V+3: PLOT H+4,V+4
360 PLOT H+3,V+5: PLOT H+2,V+6: PLOT H+1,V+7
370 PLOT H,V+8: HLIN H,H+4 AT V+9
380 RETURN
390 H = 18: V = 10
400 HLIN H,H+4 AT V+2
410 HLIN H,H+4 AT V+9
420 VLIN V+2,V+9 AT H+4
430 HLIN H+2,H+4 AT V+5
440 HLIN H+2,H+4 AT V+6
450 RETURN
460 H = 18: V = 10
470 VLIN V+2,V+9 AT H+3
480 PLOT H+2,V+3: PLOT H+1,V+4: PLOT H,V+5
490 HLIN H,H+4 AT V+6
500 RETURN
```

```
510 H = 18: V = 10
520 HLIN H,H+4 AT V+2
530 VLIN V+2,V+4 AT H
540 HLIN H,H+3 AT V+5
550 VLIN V+6,V+8 AT H+4
560 HLIN H,H+4 AT V+9
570 RETURN
```

Change the loop in line 90 to either shorten or lengthen the interval used to read switch responses. The precise time to actually complete the loop interval will depend on the size of the program and the number of statements contained within the loop.

```
90 FOR D = 1 TO 100     (about a two second interval)
90 FOR D = 1 TO 200     (about a four second interval)
90 FOR D = 1 TO 800     (about an eight second interval
```

Computational Problems

The MATH PROBLEMS program shown below illustrates how math computational skills can be presented in a single switch format. For this program, primary addition facts are displayed so that each addend is a number between 0 and 9. For each item, problems can range from 0+0 to 9+9. The problem format is easily controlled to present certain types of facts, or facts within a specific difficulty range.

After the problem has been displayed, the program automatically creates possible answers to scan, one of which is the correct answer.

```
10 REM MATH PROBLEMS
20 REM
30 HOME
40 X = INT(RND(1)*10)
50 Y = INT(RND(1)*10)
60 H = 18: IF X < 10 THEN H = 19
70 VTAB 5: HTAB H: PRINT X
80 VTAB 7: HTAB 17: PRINT "+"
90 H = 18: IF Y < 10 THEN H = 19
100 VTAB 7: HTAB H: PRINT Y
110  VTAB 8: HTAB 17: PRINT "____"
120 CA = X+Y
130 NC = 4
140 ST = 5
150 FOR L = 1 TO 750: NEXT L
160 CP = INT(RND(1)*NC+1)
170 IF CA-CP+1 > -1 THEN 190
180 CP = CP-1: GOTO 170
190 AL(CP) = CA
200 FOR K = 1 TO NC
210 IF K = CP THEN 230
220 AL(K) = CA-CP+K
230 VTAB 10+K*2: HTAB 18
240 IF AL(K) < 10 THEN PRINT " ";
250 PRINT AL(K): NEXT K
260 VL = 10
270 SP = 1
280 FOR L = 1 TO 250: NEXT L
290 VTAB VL+SP*2: HTAB 18
300 IF AL(SP) < 10 THEN PRINT " ";
310 INVERSE
320 PRINT AL(SP): NORMAL
```

```
330 FOR D = 1 TO ST*15
340 IF PEEK(-16287) > 127 THEN 420
350 IF PEEK(-16384) = 155 THEN 490
360 NEXT D
370 VTAB VL+SP*2: HTAB 18
380 IF AL(SP) < 10 THEN PRINT " ";
390 PRINT AL(SP)"  "
400 SP = SP+1: IF SP < = NC THEN 280
410 SP = 1: GOTO 280
420 IF SP = CP THEN 440
430 GOTO 470
440 VTAB VL+CP*2: HTAB 9
450 INVERSE
460 PRINT "CORRECT!";: NORMAL
470 FOR L = 1 TO 1500: NEXT L
480 GOTO 30
490 POKE -16368,0
500 END
```

For each item presented in the above program, two random numbers are generated in lines 40 and 50. The RND(1) function generates a real number greater than or equal to 0 and less than 1. In immediate execution mode, enter PRINT RND(1) and a value >=0 and <1 is displayed:

```
PRINT RND(1)
.551252583
```

If the number produced by RND(1) is .551252583, this value is multiplied by 10 (lines 40 and 50) or .551252583 X 10 = 5.51252583. The INT function then converts this value to an integer so that X is set to 5. Because the number generated by RND(1) is greater than or equal to 0 and less than 1, statements 40 and 50 produce values from 0 to 9:

By changing statements 40 and 50, the size of the addends used in each math addition fact presented can be controlled. The following illustrates different ways in which values can be confined within specific ranges:

STATEMENT	**RANGE**
INT(RND(1)*5)	0 to 4
INT(RND(1)*5+1)	1 to 5
INT(RND(1)*5+5)	5 to 9
INT(RND(1)*20)	0 to 19
INT(RND(1)*10+10)	10 to 19
INT(RND(1)*50+50)	50 to 99
INT(RND(1)*1000)	0 to 999
INT(RND(1)*11-5)	-5 to 5

Presenting the same sequence of problems each time the program is run is accomplished by beginning the program with a negative number in the RND statement. When this is done, all subsequent random numbers generated follow a specified sequence (and each negative number results in a different random number sequence):

```
25 X = RND(-12)
```

To change the MATH PROBLEMS program so that subtraction problems are displayed, lines 80 and 120 (which is used to set CA to the correct answer) are changed:

```
55 X = X + Y
80 VTAB 7: HTAB 17: PRINT "-"
```

```
120 CA = X - Y
```

Multiplication problems are created by deleting line 55 and then making these changes:

```
55
80 VTAB 7: HTAB H: PRINT "X"
120 CA = X * Y
```

As with most math problems, each problem can be displayed in various ways such as using a vertical or single line format. For division a single line format is used. To present division facts first delete lines 60 through 120 and then add the following:

```
DEL 60,120

60 CA = X
70 X = X * Y
80 VTAB 7: HTAB 15
90 PRINT X" / "Y" = "
```

Division problems are created by using the product of X and Y as the dividend and designating Y as the divisor. If X is 5 and Y is 8, then the dividend is 40 and the divisor 8 and the answer 5 (or the initial value generated by X). In this case, the line problem would be displayed as

```
40 / 8 =
```

Algebra: Just as students can develop very sophisticated single switch reading skills, a similar high level of achievement can involve mathematics. The following modify the MATH PROBLEMS program results in problems requiring the addition of signed numbers which are displayed as line problems:

```
40 X = INT(RND(1)*11-5)
50 Y = INT(RND(1)*11-5)
60
70
80
90
100 VTAB 8: HTAB 15
110 PRINT X" + "Y" = "
120 CA = X+Y
170
180
```

Open-Ended Scanning

The single switch programs presented thus far have used either a simple switch response or multiple-choice scanning routine. It is possible to enhance a scan routine to select whatever answer the student might want to enter. For example, a procedure that scans the numbers 0 to 9 can be used so that the student selects the first number of the answer, the next number, etc. until the entire answer has been selected. This open-ended format eliminates the guessing possibility associated with multiple-choice items, but the switch response required is definitely more complex than a simple scan procedure

```
10 REM OPEN-ENDED SCAN
20 REM
30 HOME
40 X = INT(RND(1)*10)
50 Y = INT(RND(1)*10)
```

```
60 H = 18: IF X < 10 THEN H = 19
70 VTAB 5: HTAB H: PRINT X
80 VTAB 7: HTAB 17: PRINT "+"
90 H = 18: IF Y < 10 THEN H = 19
100 VTAB 7: HTAB H: PRINT Y
110  VTAB 8: HTAB 17: PRINT "____"
120 CA = X+Y
130 ST = 8
140 CA$ = ""
150 S = -1
160 VTAB 17: HTAB 9
170 PRINT "R ";
180 FOR K = 0 TO 9
190 PRINT K" ";: NEXT K
200 FOR L = 1 TO 750: NEXT L
210 VTAB 17: HTAB 11+S*2
220 INVERSE: IF S > -1 THEN 250
230 VTAB 17: HTAB 9
240 PRINT "R": GOTO 260
250 PRINT S
260 NORMAL
270 FOR D = 1 TO ST*10
280 IF PEEK(-16287) > 127 THEN 380
290 IF PEEK(-16384) = 155 THEN 500
300 NEXT D
310 VTAB 17: HTAB 11+S*2
320 IF S > -1 THEN 350
330 VTAB 17: HTAB 9
340 PRINT "R": GOTO 360
350 PRINT S
360 S = S+1: IF S > 9 THEN S = -1
370 GOTO 210
380 IF S = -1 THEN 440
390 CA$ = CA$ + STR$(S)
400 VTAB 10: HTAB 1: CALL -868
410 H = LEN(CA$)
420 VTAB 10: HTAB 20-H: PRINT CA$
430 GOTO 150
440 IF CA = VAL(CA$) THEN 460
450 GOTO 480
460 VTAB 14: HTAB 15
470 PRINT "CORRECT!"
480 FOR L = 1 TO 1500: NEXT L
490 GOTO 30
500 POKE -16368,0
510 END
```

The open-ended scan procedure scans numbers 0 though 9 and stores each number selected in CA$ (see line 390). Selecting R at the beginning of the scan list signifies a RETURN and indicates that the answer selected has been entered and the answer should be evaluated. If the actual answer in CA is equal to the value in CA$ (after CA$ has been converted to a numeric value), CORRECT is printed.

To increase or decrease the amount of scan time, increase or decrease variable ST in line 130 (or adjust the timing loop in line 270). To flash the CORRECT prompt, enter the following two lines:

```
465 FLASH
475 NORMAL
```

Large Font Math

Although the programs described in this chapter can be modified using the large character set, developing math single switch programs in high-resolution graphics is not always an easy task. As is the case with reading, a large screen character set limits the amount of information that can be displayed on screen at any given time. Nonetheless, the following two programs illustrate how the large font character set can be used to generate math problems.

The FONT COUNT program is a very simple modification of the NUMBER COUNT program shown at the beginning of this chapter in that the essence of the program is to display the characters 0 through 9. Each time the switch is engaged, N is incremented by 1 and the next number in the sequence is generated. When the program is first run, N is first set to 48 which corresponds to the LASCII shape **0**. After N has been incremented to the highest value or 57 and **9** displayed, N is re-set to 48 and the number sequence is repeated.

Because this is a very easy program to use and understand, experiment with the program by modifying the HCOLOR statement in line 110 and the shape position values (which are currently set to 125 and 75) in line 130.

```
10 REM FONT COUNT
20 REM
30 HOME
40 PRINT CHR$(4);"BLOAD LASCII"
50 A = PEEK(43634) + PEEK(43635) * 256
60 T = INT(A/256)
70 POKE 232,A-T*256
80 POKE 233,T
90 N = 48
100 HGR
110 HCOLOR=7
120 SCALE=1
130 DRAW N AT 125,75
140 N = N+1: IF N > 57 THEN N = 48
150 IF PEEK(-16287) > 127 THEN 150
160 IF PEEK(-16287) > 127 THEN 100
170 IF PEEK(-16384) = 155 THEN 190
180 GOTO 160
190 POKE -16368,0
200 TEXT: HOME
210 END
```

The FONT MATH PROBLEMS program generates a series of up to N addition fact problems. As with the MATH PROBLEMS, X and Y are generated as the problem addends. The position of the correct answer is determined by the variable CP (line 160), and the incorrect alternatives are then generated.

The problem is displayed by first transforming the problem to a string value in line 280 and then calling the character display subroutine beginning in line 310. The alternatives are also displayed by transforming each of the possible answers to the string variable P$ (line 410) and then calling the character display routine in 310.

```
10 REM FONT MATH PROBLEMS
20 REM
30 HOME
40 N = 5
50 NC = 4
60 ST = 5
```

```
70 PRINT CHR$(4);"BLOAD LASCII"
80 A = PEEK(43634) + PEEK(43635) * 256
90 T = INT(A/256)
100 POKE 232,A-T*256
110 POKE 233,T
120 FOR MP = 1 TO N
130 X = INT(RND(1)*10)
140 Y = INT(RND(1)*10)
150 CA = X+Y
160 CP = INT(RND(1)*NC+1)
170 IF CA-CP+1 > -1 THEN 190
180 CP = CP -1: GOTO 170
190 AL(CP) = CA
200 FOR K = 1 TO NC
210 IF K = CP THEN 230
220 AL(K) = CA-CP+K
230 NEXT K
240 NA = NA+1
250 HGR: HCOLOR=7
260 SCALE=1
270 V = 15: H = 70
280 P$ = STR$(X) + " + " + STR$(Y) + " = "
290 GOSUB 310
300 GOTO 380
310 FOR L = 1 TO LEN(P$)
320 C$ = MID$(P$,L,1)
330 C = ASC(C$)
340 DRAW C AT H,V
350 H = H+13
360 NEXT L
370 RETURN
380 V = 25
390 FOR K = 1 TO NC
400 V = V+25: H = 100
410 P$ = STR$(AL(K)): GOSUB 310
420 NEXT K
430 POKE -16368,0
440 V = 50: H = 75
450 SP = 1: NS = 1
460 HCOLOR=7
470 DRAW 31 AT H,V
480 FOR D = 1 TO ST*15
490 IF PEEK(-16287) > 127 THEN 560
500 IF PEEK(-16384) = 155 THEN 670
510 NEXT D
520 HCOLOR=0: DRAW 31 AT H,V
530 V = V+25: IF V > 25+25*C THEN V = 50
540 SP = SP+1: IF SP > NC THEN SP = 1
550 GOTO 460
560 V = 25+SP*25: H = 100
570 P$ = STR$(AL(SP))
580 HCOLOR=0: GOSUB 310
590 H = 115: HCOLOR=7: GOSUB 310
600 IF SP = CP THEN 620
610 GOTO 650
620 P$ = "CORRECT!"
630 H = 155: GOSUB 310
640 SC = SC+1
650 FOR D = 1 TO 2000: NEXT D
660 NEXT MP
670 POKE -16368,0
680 TEXT: HOME
```

```
690 VTAB 7
700 PRINT "NUMBER = "N
710 PRINT "ATTEMPTED = "NA
720 PRINT "CORRECT = "SC
730 END
```

The number of problems presented is determined by N in line 40 and the number of alternatives per problem by NC in line 50. To display five alternatives, use HGR2 in order use the entire high-resolution graphics screen:

```
50 NC = 5
250 HGR2: HCOLOR=7
```

Subtraction: To present subtraction fact problems, the minuend is the sum of X and Y and the subtrahend is Y so that each problem has the form (X+Y) - Y = X. ^The minuend is first set to X+Y in line 145, the correct answer to X-Y in line 150, and the subtraction form of the problem to P$ in line 280:

```
145 X = X+Y
150 CA =X-Y
280 P$ = STR$(X) + " - " + STR$(Y) + " = "
```

Be sure to insert the quotation marks as shown. In line 280 the entire problem is converted to a string. The process of adding the four separate string parts (the sum of X and Y, the minus sign, the subtrahend and the equal sign) is called concatenation which entails adding strings together using the plus sign.

Following an incorrect response, no feedback is given. To indicate that a response is incorrect, an **X** can be placed in front of the incorrect alternative selected:

```
605 HCOLOR=0: DRAW 31 AT 75,V
606 HCOLOR=7: DRAW 88 AT 75,V
```

Additional information following an incorrect response (e.g., the correct answer) could be provided as deemed appropriate. A routine could also be added to first provide correct answer feedback following an incorrect response, and then re-presenting the problem. As is the case with all single switch programs, specific student needs must be considered when designing various program options.

Multiplication: Delete line 45 if the above subtraction format has been used, set CA to the product of X and Y in line 30, and modify line 280.

```
145
150 CA = X*Y
280 P$ = STR$(X) + "X" + STR$(Y) + " = "
```

Division: Division problems are generated with no remainders by setting the answer to Y, the dividend to the product of X and Y, and the divisor to X. If X (i.e., the divisor) is 0, a new X value is generated (line 135).

```
135 IF X = 0 THEN 130
150 CA = Y
280 P$ = STR$(X*Y) + " / " + STR$(X) + "="
```

The use of graphics allows the incorporation of symbols not readily available with normal text symbols. Problems can be displayed using the symbol √ by making these modifications:

```
280 P$ = STR$(X) + " " + STR$(X*Y)
331 IF C = 32 THEN 333
332 GOTO 340
333 HPLOT H+5,10 TO H+35,10
334 HPLOT H,29 TO H+5,10
336 GOTO 350
```

or the ÷ symbol could be used by making these modifications:

```
280 P$ = STR$(X*Y) + " " +STR$(X) + "="
333 HPLOT H-6,22 TO H+4,22
334 HPLOT H-2,20 TO H,20
335 HPLOT H-2,24 TO H,24
```

Single Switch Math Systems

By integrating several components within a program, a program system can be developed that provides considerable program use flexibility. The following two programs, MATHSCAN and SWITCH CALCULATOR, integrate addition, subtraction, multiplication and division activities in a single switch format.

Single Switch Primary Math Facts

MATHSCAN can be used to provide computational practice in each of the major operations using a single switch format. When this program is run, three values must be entered: scan speed, problem type, and difficulty level. If the RETURN key is pressed after one or all of the input prompts, the program sets the control variables to the built-in default values.

The number of items presented for each task is determined by variable N in line 230. The current value of N is 10. The program is set so that the same item is not repeated (e.g., if **3 X 4** is presented, **4 X 3** is not presented during the same task). Because of this, N cannot be set higher than 15 for addition, subtraction, and multiplication. For division, N cannot be set higher than 10. The matrix M (see line 30) is used to insure that the same problem is not presented twice during the same task.

The number of alternatives is set by SA in line 220. To change the number of alternatives presented with each item to 3, the following modification would be made:

```
220 SA = 3
```

The number of alternatives comprising each list can range from 2 to as many as 6. For each item presented, the program uses a randomization routine when creating and displaying alternatives.

```
10 REM MATHSCAN
20 REM
30 DIM M(4,99)
40 HOME
50 VTAB 2: HTAB 16
60 PRINT "MATHSCAN"
70 VTAB 22: HTAB 3
80 PRINT "(PRESS RETURN AFTER EACH DATA ENTRY)"
90 VTAB 5
100 INPUT "SCAN SPEED (1=FAST TO 10=SLOW): ";SP$
```

```
110 IF SP$ < "1" THEN SP$ = "5"
120 SP = VAL(SP$)
130 PRINT
140 INPUT "PROB (1=ADD, 2=SUB, 3=MUL, 4=DIV): ";T$
150 T = VAL(T$): IF T < 1 OR T > 4 THEN T = 1
160 PRINT
170 INPUT "DIFFICULTY (1=EASY, 2=AVE, 3=DIF): ";D$
180 D = VAL(D$): IF D < 1 OR D > 3 THEN D = 1
190 R = RND(-PEEK(78)*100+PEEK(79))
200 S$(1) = "+": S$(2) = "-": S$(3) = "X"
210 HOME
220 SA = 4
230 N = 10
240 CN = CN+1: IF CN > N THEN 1060
250 IF D = 2 THEN 300
260 IF D = 3 THEN 360
270 X = INT(RND(1)*5)
280 Y = INT(RND(1)*5)
290 GOTO 380
300 X = INT(RND(1)*10)
310 Y = INT(RND(1)*10)
320 IF D < > 2 THEN 380
330 IF X < 5 AND Y < 5 THEN 300
340 IF X > 4 AND Y > 4 THEN 300
350 GOTO 380
360 X = INT(RND(1)*5+5)
370 Y = INT(RND(1)*5+5)
380 IF M(T,X*10+Y) = 1 THEN 250
390 IF M(T,Y*10+X) = 1 THEN 250
400 M(T,X*10+Y) = 1
410 M(T,Y*10+X) = 1
420 IF T = 4 AND Y = 0 THEN 250
430 IF Y = 0 THEN 250
440 IF T = 1 THEN CA = X+Y
450 IF T = 2 OR T = 4 THEN CA = X
460 IF T = 2 THEN X = X+Y
470 IF T = 3 THEN CA = X*Y
480 HOME
490 HC = 19
500 IF T = 2 AND X > 9 THEN HC = HC-1
510 IF T = 4 THEN 560
520 VTAB 8: HTAB HC: PRINT X
530 VTAB 9: HTAB 17: PRINT S$(T)
540 VTAB 10: HTAB 17: PRINT "----"
550 GOTO 580
560 VTAB 8: HTAB 17: PRINT "-----"
570 VTAB 9: HTAB 16: PRINT Y") "X*Y
580 FOR L = 1 TO 250: NEXT L
590 CP = INT(RND(1)*SA+1)
600 IF CA-CP+1 > -1 THEN 620
610 CP = CP-1: GOTO 600
620 AL(CP) = CA
630 FOR K = 1 TO SA
640 IF K = CP THEN 660
650 AL(K) = CA-CP+K
660 VTAB 10+K*2: HTAB 18
670 IF AL(K) < 10 THEN PRINT SPC(1)
680 PRINT AL(K): NEXT K
690 VL = 10: SL = 1
700 FOR L = 1 TO 250: NEXT L
710 VTAB VL+SL*2: HTAB 18
720 IF AL(SL) < 10 THEN PRINT SPC(1)
```

```
730 INVERSE
740 PRINT AL(SL): NORMAL
750 FOR ST = 1 TO SP*10
760 IF PEEK(-16286) > 127 OR PEEK(-16287) > 27 THEN 850
770 KY = PEEK(-16384) : IF KY > 127 THEN 850
780 IF PDL(0) < 20 THEN 850
790 NEXT ST
800  VTAB VL+SL*2: HTAB 18
810 IF AL(SL) < 10 THEN PRINT SPC(1)
820 PRINT AL(SL)"  "
830 SL = SL+1: IF SA >= SL THEN 700
840 SL = 1: GOTO 700
850 POKE -16368,0
860 IF SL = CP THEN 970
870 VTAB VL+SL*2: HTAB 18
880 IF AL(SP) < 10 THEN PRINT SP(1)
890 PRINT AL(SL)"  "
900 VTAB VL+CP*2: HTAB 18
910 PRINT "----->";
920 HTAB 18: IF CA < 10 THEN PRINT SPC(1)
930 INVERSE
940 PRINT CA;: NORMAL: PRINT "  "
950 FOR L = 1 TO 1000: NEXT L
960 GOTO 1030
970 VTAB VL+CP*2: HTAB 9
980 INVERSE
990 PRINT "CORRECT!";: NORMAL
1000 CO = CO+1
1010 FOR L = 1 TO 2500: NEXT L
1020 CN = CN+1
1030 FOR L = 1 TO 1000: NEXT L
1040 IF KY = 155 THEN 1060
1050 GOTO 240
1060 HOME: VTAB 5
1070 PRINT "RESULTS:"
1080 PRINT: PRINT "NUMBER OF ITEMS = "CN
1090 PRINT "NUMBER CORRECT = "CO
1100 PRINT "PERCENT CORRECT = "INT(CO/CN*100+.5)
1110 END
```

Switch Calculator

As is the case with language boards, matrix scanning can be used in conjunction with single switch math programs. The SWITCH CALCULATOR program displays a selection matrix comprised of four rows and four columns:

1	2	3	4
5	6	7	8
9	0	+	-
X	/	.	=

The scanning process is twofold: 1) Each row is scanned by highlighting the rows in sequential order until a selection is made; 2) after a row has been selected, each row element is scanned until a selection is made. Following each value entered, an operation is selected: addition (+), subtraction (-), multiplication (X), or division (/). After the second number has been entered, the = symbol is used to display the answer.

Before the SWITCH CALCULATOR screen is displayed, the scan speed is entered. Begin using the program with simple problems (e.g., 2 + 1 =). Multiple digit numbers can be used by simply selecting digits from the matrix before selecting the problem operation.

```
10 REM SWITCH CALCULATOR
20 REM
30 S$(1) = "1  2  3  4"
40 S$(2) = "5  6  7  8"
50 S$(3) = "9  0  +  -"
60 S$(4) = "X  /  .  ="
70 FOR K = 1 TO 17: H$ = H$ + "-": NEXT
80 SM = 10
90 HOME
100 VTAB 2: HTAB 12
110 PRINT "SWITCH CALCULATOR"
120 VTAB 22: HTAB 3
130 PRINT "(ENTER SCAN SPEED AND PRESS RETURN)"
140 VTAB 9: HTAB 1
150 INPUT "SCAN SPEED (1=FAST, 10=SLOW): ";ST$
160 IF ST$ < "1" THEN ST$ = "1"
170 ST = VAL(ST$)
180 VTAB 3: HTAB 1: CALL -958
190 VTAB 4: HTAB 12: PRINT H$
200 VTAB 12: HTAB 12: PRINT H$
210 POKE 35,10
220 VX = 5: HX = 23
230 VTAB VX: HTAB HX
240 GOSUB 500
250 IF KY = 195 OR KY = 227 THEN 220
260 AN = VAL(N$)
270 O$ = C$
280 PRINT: HTAB 13
290 PRINT O$;: HTAB 23
300 VX = PEEK(36)+1: HX = PEEK(37)+1
310 GOSUB 500
320 IF KY = 195 OR KY = 227 THEN 220
330 VTAB VX: HTAB HX
340 IF N$ = "" THEN 270
350 B$ = N$: B = VAL(B$)
360 IF O$ = "+" THEN AN = AN+B
370 IF O$ = "-" THEN AN = AN-B
380 IF O$ = "X" THEN AN = AN*B
390 IF O$ = "/" THEN AN = AN/B
400 X = 0: IF C$ = "=" THEN X = 3: GOTO 420
410 GOTO 270
420 VTAB VX+1: HTAB 13
430 PRINT C$;
440 A$ = STR$(AN): N$ = A$
450 O$ = C$
460 GOSUB 680
470 VX = VX+X: VTAB VX
480 IF C$ = "=" THEN 220
490 GOTO 310
500 N$ = "": M=0
510 HX = PEEK(36)+1: VX = PEEK(37)+1
520 IF C$ = "=" THEN EC = 1
530 GOSUB 770
540 IF KY = 195 OR KY = 227 THEN RETURN
550 VTAB VX: HTAB HX
560 IF C$ = "+" OR C$ = "-" OR C$ = "X" THEN RETURN
    570 IF C$ = "/" OR C$ = "=" THEN RETURN
```

```
580 N$ = N$+C$
590 IF EC = 0 THEN 640
600 FOR K = 5 TO 11: VTAB K: HTAB 1
610 CALL -868: NEXT K
620 EC = 0
630 VX = 5: VTAB VX: HTAB HX
640 IF LEN(N$) > 8 THEN N$ = LEFT$(N$,8)
650 L = LEN(N$)
660 GOSUB 680
670 GOTO 530
680 FOR L = 1 TO LEN(N$)
690 L$ = MID$(N$,L,1)
700 IF L$ = "." THEN M = L
710 NEXT L: IF M > 0 THEN 730
720 HTAB 22-LEN(N$): GOTO 740
730 HTAB 23-M
740 PRINT N$;
750 IF C$ = "=" THEN PRINT
760 RETURN
770 POKE 35,23: VP = 1
780 FOR J = 1 TO 4
790 VTAB J*2+12: HTAB 15
800 PRINT S$(J): NEXT J
810 INVERSE
820 VTAB 14: HTAB 15
830 PRINT S$(VP)
840 NORMAL
850 GOSUB 1170
860 IF KY = 195 OR KY = 227 THEN RETURN
870 IF SW = 1 THEN SW = 0: GOTO 950
880 VTAB VP*2+12: HTAB 15
890 PRINT S$(VP)
900 VP = VP+1: IF VP > 4 THEN VP = 1
910 VTAB VP*2+12: HTAB 15: INVERSE
920 PRINT S$(VP)
930 NORMAL
940 GOTO 850
950 VTAB VP*2+12: HTAB 15
960 PRINT S$(VP)
970 INVERSE
980 VTAB VP*2+12: HTAB 15
990 PRINT MID$(S$(VP),1,1)
1000 NORMAL
1010 HP = 15
1020 GOSUB 1170
1030 IF KY = 195 OR KY = 227 THEN RETURN
1040 VTAB VP*2+12: HTAB HP
1050 PRINT MID$(S$(VP),HP-14)
1060 IF SW = 1 THEN 1130
1070 HP = HP+3: IF HP > 24 THEN HP = 15
1080 VTAB VP*2+12: HTAB HP
1090 INVERSE
1100 PRINT MID$(S$(VP),HP-14,1)
1110 NORMAL
1120 GOTO 1020
1130 SW = 0
1140 C$ = MID$(S$(VP),HP-14,1)
1150 POKE 34,7: POKE 35,18
1160 RETURN
1170 FOR D = 1 TO ST*SM
1180 IF PDL(0) < 20 THEN 1220
1190 IF PEEK(-16286) > 127 OR PEEK(-16287) > 127 THEN
```

```
     1220
1200 KY = PEEK(-16384): IF KY > 127 THEN 1220
1210 NEXT D: SW = 0:RETURN
1220 POKE -16368,0: SW = 1
1230 FOR D = 1 TO 300 + (ST*SM*7): NEXT D
1240 IF KY = 155 THEN 1320
1250 IF KY = 195 OR KY = 227 THEN 1270
12602 GOTO 1310
1270 FOR J = 5 TO 11: VTAB J: HTAB 1
1280 CALL -868: NEXT J: GOTO 1300
1290 GOTO 1300
1300 SW = 0
1310 RETURN
1320 POKE 34,0: POKE 35,24: HOME
1330 END
```

Chapter 8

Single Switch Reading

From Readiness to Reading

When an individual has demonstrated the ability to discriminate, the time is right to begin experimenting with reading readiness. The term "experimenting" is not used in a research vain, but rather to emphasize that many different features regarding scan routines must be considered in order to find those factors which best meet a specific individual's learning needs. The factors to be considered include the type of scan, scan speed, item, presentation (e.g., presenting items tachistoscopically) and feedback.

The LETTER RECOGNITION program described below illustrates many of the modifications that can be made to enhance a single switch scan program. This easy-to-use program can be modified to develop a variety of matching and short term memory skills. If the student does not understand the task or task content, use one of the low-resolution scan routines presented in Chapter 3.

```
10 REM LETTER RECOGNITION
20 REM
30 A = 4
40 FOR X = 1 TO 7
50 HOME
60 FOR K = 1 TO A
70 S$(K) = CHR$(RND(1)*26+65)
80 FOR L = 0 TO K-1
90 IF S$(K) = S$(L) THEN 70
100 NEXT L
110 VTAB 7+K*2: HTAB 19
120 PRINT S$(K)
130 NEXT K
140 CP = INT (RND(1)*A+1)
150 VTAB 5: HTAB 19
160 PRINT S$(CP)
170 INVERSE
180 VTAB 9+VP: HTAB 15
190 PRINT "  ": NORMAL
200 FOR K = 1 TO 100
210 IF PEEK(-16287) > 127 THEN 280
220 NEXT K
230 VTAB 9+VP: HTAB 15
240 PRINT "  "
250 VP = VP+2
260 IF VP > A*2-2 THEN VP = 0
270 GOTO 170
280 IF CP = VP/2+1 THEN 300
290 GOTO 320
300 VTAB 20: HTAB 15
310 PRINT "CORRECT!"
320 FOR L = 1 TO 2000: NEXT L
330 VP = 0
340 NEXT X
```

For each item presented, four letters are randomly generated and stored in variables S$(1), S$(2), S$(3), and S$(4). The loop in lines 80-100 insures that all four letters are different. One of the four letters is designated as correct response (see line 140).

After the four alternatives are displayed, each alternative is scanned. If the switch is engaged while an alternative is being scanned, that alternative is evaluated as student's selected answer. If the alternative selected matches the stimulus, a CORRECT prompt is shown. If the alternative does not match the stimulus, no feedback is given.

The program can be modified to include control variables to specify the number of items and scan speed. The number of items can be preset by

```
31 HOME: VTAB 7
32 INPUT "NUMBER OF ITEMS? ";N
40 FOR X = 1 TO N
```

The scan speed can be specified by adding these lines:

```
33 VTAB 7
34 INPUT "SCAN SPEED (1=FAST, 9=SLOW)? ";SP
200 FOR K = 1 TO 35*SP
```

If the task is too difficult, we can limit the number of choices by changing variable A in line 30. In order to present three alternatives rather than four, reset variable A in line 30:

```
30 A = 3
```

As the program is now written, a solid bloc cursor is used to scan each alternative. This is accomplished by displaying two "blank" spaces in inverse mode. To change the scan type, make the following modification:

```
190 PRINT "->": NORMAL
```

Each of the alternatives can be scanned by using alternative numbers. The following scans each alternative using 1, 2, 3 and 4 in sequential order:

```
190 PRINT VP/2+1: NORMAL
```

Or letters can be used as the cursor:

```
190 PRINT CHR$(65+VP/2)
```

Very often a small or subtle change can give a program a better feel. For example, when the cursor moves from one alternative to the next, there is no delay. A short delay is accomplished by inserting a delay loop in line 225:

```
225 FOR D = 1 TO 100: NEXT D
```

Line 70 indicates the type of character ASCII character generated. This statement can be modified to present numbers, lowercase letters, or keyboard symbols.

Change line 70 to display numbers 0 to 9 by

```
70 S$(K) = CHR$(RND(1)*10+48)
```

Or change the line to display lowercase letters by

```
70 S$(K) = CHR$(RND(1)*26+97)
```

Finally, set variable A in line 30 to 5, and then change line 70 to display the whole array of keyboard symbols (and thus create a fairly difficult matching task):

```
30 A = 5
70 S$(K) = CHR$(RND(1)*95+33)
```

Coding

The concept underlying the LETTER RECOGNITION program can be used to create a coding task that is often used in psychological assessment. The student is given the code such as 1=*, 2=+, 3=#, and when the stimulus symbol is shown, the student enters the corresponding response symbol. If 1 is shown, the student enters * (or whatever corresponds to 1).

The CODING program creates a new code each time the program is run. The following is an example of the type of code that might be generated:

)	/	#	+	$	V	-	*	(
1	2	3	4	5	6	7	8	9

For each item presented, a stimulus symbol is randomly selected and displayed. Next, four alternatives are listed and scanned from left to right. The task is to select the alternative that matches the stimulus code which is displayed on the screen throughout the task.

The CODING task continues until the Esc key is pressed.

```
10 REM CODING
20 REM
30 SP = 10
40 NC = 4
50 FOR K = 1 TO 9
60 R$(K) = STR$(K)
70 READ V$(K): NEXT K
80 DATA *,),+,-,#,V,(,$,/
90 FOR K = 1 TO 9
100 R = INT(RND(1)*9+1)
110 IF V(R) = 1 THEN 100
120 V(R) = 1
130 S$(K) = V$(R)
140 NEXT K
150 HOME
160 N = N+1
170 FOR L = 1 TO 1000: NEXT L
180 VTAB 3: HTAB 18
190 PRINT "CODING"
200 FOR K = 1 TO 9
210 VTAB 7: HTAB 6+K*3
220 PRINT S$(K)
230 VTAB 9: HTAB 6+K*3
240 PRINT R$(K)
250 NEXT K
260 R = INT(RND(1)*9+1)
270 CR$ = R$(R)
280 VTAB 15: HTAB 9
290 PRINT CR$"  = ";
300 FOR L = 1 TO 9
310 A(L) = 0: S(L) = 0: NEXT L
320 RS = INT(RND(1)*NC+1)
330 A$(RS) = S$(R)
340 FOR K = 1 TO NC
350 RN = INT(RND(1)*9+1)
```

```
360 IF K = RS THEN 400
370 A$(K) = S$(RN)
380 IF A$(K) = S$(R) THEN 350
390 IF S(RN) = 1 THEN 350
400 S(RN) = 1
410 PRINT "  ";A$(K)"  ";
420 NEXT K: PRINT
430 NK = 1
440 VTAB 16: HTAB 12+NK*5
450 INVERSE: PRINT " "
460 NORMAL
470 FOR D = 1 TO SP*10
480 IF PEEK(-16287) > 127 550
490 IF PEEK(-16384) = 155 THEN 630
500 NEXT D
510 VTAB 16: HTAB 12+NK*5
520 PRINT " "
530 NK = NK+1: IF NK > NC THEN 430
540 GOTO 440
550 IF NK = RS THEN 570
560 GOTO 610
570 C = C+1
580 VTAB 20: HTAB 17
590 FLASH: PRINT "CORRECT!"
600 NORMAL
610 FOR L = 1 TO 2500: NEXT L
620 GOTO 150
630 VTAB 20
640 PRINT "NUMBER = "N-1
650 PRINT "CORRECT = "C
660 END
```

The scan speed is determined by variable SP in line 30, and the number of alternatives comprising the alternative list for each item is determined by NC in line 40. The number of alternatives for items can be set to either 2, 3, 4 or 5.

The response symbols that correspond to the nine stimulus symbols are contained in the DATA statement in line 80:

```
80 DATA *,),+,-,#,V,(,$,/
```

To change the symbols to letters, modify line 80:

```
80 DATA A,B,C,D,E,F,G,H,I
```

Word Recognition

Single switch programs can be created using many different reading, writing and/or language arts activities. The READ program described in this section is useful for developing sight-word recognition skills and to review content area vocabulary. Because of the space required to display letters in graphics mode, this program is presented in normal text screen format.

The READ program operates by first reading up to 50 words from DATA statements. The program continues to read words until an error occurs at which point the ONERR statement in line 80 branches to line 120. This routine causes the program to automatically read the DATA statements, and to count the number of items (as indicated by the variable N) read from DATA statements

After the N words have been read, one of the words from the W$ vector

BEST COPY AVAILABLE

containing the words is selected and displayed on screen. For each word displayed, four alternatives are selected from other words in the word pool. The task is to engage the switch when the target word displayed at the top of the screen is scanned in the word list. The task continues until all the words have been presented.

The number of alternatives presented for each task item is determined by variable AL in line 60. The number of alternatives should be set to a value from 2 to as many as 8. To increase or decrease the length of time each alternative is scanned, increase or decrease the value of ST in line 70. The following is an example of a target word and four alternatives:

```
  LAMP

  PENCIL
  RADIO
  SNOW
█ LAMP
```

The variable RW is used to select a word from the word pool, and CP signifies the position of the correct alternative. The W(RW) vector in lines 150 and 160 insures that each word appears no more than once for each task run.

The type of cursor used is changed by modifying variable CR$ in line 50. As the program is now written, CR$ is set to two blank spaces. This is shown in line 50 as

```
50 CR$ = "  "
```

or

```
50 CR$ = CHR$(32) + CHR$(32)
```

To change the cursor in CR$ to key board symbols, reset CR$ in line 50, and then change line 390 as shown:

```
50 CR$ = ">>"
390 PRINT "  "
```

To use more than 50 words in the program, change the dimension statements in line 30. To present the words in sequential order, set variable RW to W in line 140 and delete lines 150 and 160.

```
10 REM READ
20 REM
30 DIM W$(50),W(50),A(50)
40 HOME
50 CR$ = "  "
60 AL = 4
70 ST = 5
80 ONERR GOTO 120
90 N = 1
100 READ W$(N)
110 N = N+1: GOTO 100
120 N = N-1: POKE 216,0
130 FOR W = 1 TO N
140 RW = INT(RND(1)*N+1)
150 IF W(RW) = 1 THEN 140
160 W(RW) = 1
```

```
170 HOME: VL = 7
180 CP = INT(RND(1)*AL+1)
190 A$(CP) = W$(RW)
200 VTAB 5: HTAB 16
210 PRINT A$(CP)
220 FOR K = 1 TO AL
230 IF K = CP THEN 280
240 RN = INT(RND(1)*N+1)
250 IF A(RN) = 1 OR RN = RW THEN 240
260 A(RN) = 1
270 A$(K) = W$(RN)
280 VTAB VL+K*2: HTAB 16
290 PRINT A$(K)
300 NEXT K
310 P = 1
320 INVERSE
330 VTAB VL+P*2: HTAB 13
340 PRINT CR$: NORMAL
350 FOR D = 1 TO ST*30
360 IF PEEK(-16287) > 127 THEN 420
370 NEXT D
380 VTAB VL+P*2: HTAB 13
390 PRINT CR$
400 P = P+1: IF P < = AL THEN 320
410 P = 1: GOTO 320
420 IF P = CP THEN 440
430 GOTO 470
440 VTAB VL+CP*2: HTAB 6
450 INVERSE
460 PRINT "CORRECT!": NORMAL
470 FOR L = 1 TO 2000: NEXT L
480 FOR K = 1 TO N: A(K) = 0: NEXT K
490 NEXT W
500 DATA  HAMBURGER,MILK,PLAY,SNOW,
    TELEVISION,RADIO,ORANGE,PENCIL.LAMP,
    Florida
```

DATA Statements

The words used in the READ program are contained in a DATA statement in line 500. These words should be modified to include words of interest to the student. The following modification illustrates how 10 different words can be used with the program:

```
500 DATA MRS. SMITH, basketball, Springfield,
    bus,Giant Food, store, Main Street, Pepsi,

    Room 324,McDonald's
```

If desired, and if a large number of words are being used, several DATA statements can be used:

```
500 DATA Mrs. Smith, basketball, Springfield
510 DATA bus, Giant Food, store
520 DATA Main Street, Pepsi, Room 324, McDonald's
```

The only rule to remember is that each data statement must begin with the word DATA.

Instructional Feedback

Although it is possible to present items in what is sometimes referred to as "test mode" in that no feedback is given whatsoever, feedback following a correct response for single switch users is almost always warranted.

However, what type of feedback should be given following an incorrect response? It is certainly safe to say that sad faces and similar types of feedback probably does nothing more than to reinforce incorrect responses. On the other hand, it might be useful *on some occasions* following an incorrect response to highlight the correct answer. The following modifications result in the alternatives displayed on screen being cleared and then the correct alternative displayed following an incorrect response.

```
421 VTAB 6: CALL -958
422 FOR L = 1 TO 500: NEXT L
423 VTAB VL+CP*2: HTAB 16
424 PRINT A$(CP)
```

Although this is a matter of personal instructional preference and individual need, instructional feedback seems to be most useful for older students. Some younger students, on the other hand, might not make the connection between the problem and correct answer (e.g., a student might be unconcerned whether a correct or incorrect alternative is selected because the correct answer is displayed following an incorrect response).

Tachistoscopic Techniques

2 A tachistoscope presents stimuli or a stimulus for a specified period of time. The READ program is changed to present items tachistoscopically by adding a time delay loop following the presentation of the stimulus word and then clearing the line on which the stimulus word is displayed:

```
215 FOR L = 1 TO 1500: NEXT L
216 VTAB 5: CALL -868
```

The CALL -868 statement clears screen line 5 where the stimulus item is displayed.

If desired, after the stimulus word is cleared from the screen, the first letter of the stimulus word can be provided as a clue:

```
217 VTAB 5: HTAB 16
218 PRINT LEFT$(A$(CP),1)
```

The LEFT$ statement is used to print the left-most character of the correct answer string. A further clue could be given by showing the number of letters contained in the words by a series of dashes. Be sure to include the punctuation exactly as shown when making this modification:

```
218 PRINT LEFT$(A$(CP),1);
219 FOR L = 1 TO LEN(A$(CP))-1: PRINT "-";: NEXT L
```

Large Font Scanning

The programs described in this section can be modified for use with the large font character set LASCII. However, remember that the program will require considerable modification to accommodate the large screen characters. As mentioned previously, if a student can use the normal video display mode, this should be used.

The SCAN LETTER FONT program first displays a letter using the large character set. Next, a series of alternatives are displayed and scanned sequentially. The number of alternatives is set in line 50 and can vary from 1 to 6, where a setting of 1 is basically a cause/effect program:

```
10 REM SCAN LETTER FONT
20 REM
30 DIM W$(26)
40 D$ = CHR$(4): G$ = CHR$(5)
50 WP = 5
60 N = 10
70 HOME
80 VTAB 2: HTAB 14: PRINT "LETTER FONT"
90 VTAB 18
100 PRINT "(PRESS RETURN TO BEGIN)";
110 POKE -16368,0: GET R$
120 IF ASC(R$) = 27 THEN VTAB 23: END
130 PRINT
140 HOME
150 PRINT D$;"BLOAD LASCII"
160 A = PEEK(43634) + PEEK(43635) * 256
170 T = INT(A/256)
180 POKE 232,A-T*256
190 POKE 233,T
200 VTAB 3: HTAB 16: PRINT "LETTER FONT"
210 POKE 216,0: VTAB 22: HTAB 1
220 PRINT "(ENTER CONTROL VALUES AND PRESS RETURN)"
230 VTAB 9: HTAB 1
240 INPUT "SCAN SPEED (1=FAST, 10=SLOW): ";ST$
250 ST = VAL(ST$): IF ST < 1 THEN ST = 1
260 PRINT
270 PRINT "TACHISTOSCOPE (1=ON, 2=OFF): ";TC$
280 POKE -16368,0: HOME
290 FOR X = 1 TO N
300 RW = INT(RND(1)*26)
310 NA = NA+1
320 HGR2: HCOLOR=7
330 SCALE=1
340 P$(1) = CHR$(65+RW)
350 FOR K = 2 TO WP
360 RN = INT(RND(1)*26)
370 IF RW = RN THEN 360
380 P$(K) = CHR$(65+RN)
390 NEXT K
400 V = 10: H = 125
410 W$ = P$(1): GOSUB 450
420 V = 24: H = 125
430 W$ = "-": GOSUB 450
440 GOTO 520
450 FOR L = 1 TO LEN(W$)
460 C$ = MID$(W$,L,1)
470 C = ASC(C$)
480 DRAW C AT H,V
490 H = H+13
500 NEXT L
510 RETURN
520 IF TC$ = "2" THEN 560
530 FOR D = 1 TO 500*ST: NEXT D
540 HGR2
550 HCOLOR=7
560 V = 25
```

```
570 FOR WA = 1 TO WP
580 RN = INT(RND(1)*WP+1)
590 IF A(RN) = 1 THEN 580
600 A(RN) = 1: PC(WA) = RN
610 IF RN = 1 THEN CP = WA
620 V = V+25: H = 125
630 W$ = P$(RN): GOSUB 450
640 NEXT WA
650 POKE -16368,0
660 V = 50: H = 100
670 SP = 1: NS = 1
680 HCOLOR=7
690 DRAW 31 AT H,V
700 FOR D = 1 TO ST*10
710 IF PEEK(-16287) > 127 THEN 780
720 KY = PEEK(-16384): IF KY > 127 THEN 780
730 NEXT D
740 HCOLOR=0: DRAW 31 AT H,V
750 V = V+25: IF V > 25+25*WP THEN V = 50
760 SP = SP+1: IF SP > WP THEN SP = 1
770 GOTO 680
780 POKE -16368,0
790 IF KY = 155 THEN 940
800 V = 25+SP*25: H = 125
810 W$ = P$(PC(SP))
820 HCOLOR=0: GOSUB 450
830 H = 145: HCOLOR=7: GOSUB 450
840 FOR D = 1 TO 750: NEXT D
850 IF SP = CP THEN 870
860 GOTO 890
870 W$ ="*--->": H = 20: GOSUB 450
880 SC = SC+1
890 POKE -16368,0
900 FOR D = 1 TO 2000: NEXT D
910 FOR K = 0 TO WP
920 A(K) = 0: NEXT K
930 NEXT X
940 TEXT: HOME
950 VTAB 7
960 PRINT "NUMBER = "N
970 PRINT "ATTEMPTED = "NA
980 PRINT "CORRECT = "SC
990 END
```

The following statements illustrate how a program is easily modified for use with an Echo. The amount of feedback presented via the Echo can be expanded by modifying the PRINT statement in line 875.

```
145 PRINT D$;"BRUN TEXTALKER"
146 HOME: PRINT G$;"O"
435 PRINT G$;"T"
436 PRINT P$(1)
685 PRINT P$(PC(SP))
875 PRINT "CORRECT"
```

The SCAN WORD FONT program is similar to the SCAN LETTER FONT program in terms of the multiple choice format. However, this program has been written to work in conjunction with an Echo. After the TEXTALKER program has been run, the value 99 is placed in register 999. If the program is interrupted, and then re-run, the statement to run the TEXTALKER is by-passed because the program is already in memory. The same technique is used with the LASCII character set. Because the values in PEEK(999) and PEEK(1000) will not change (usually) when another program is run, the need to run either the TEXTALKER and load LASCII is not required each time a program is run.

The number of alternatives is determined by setting WP to a value from 1 to 5 in line 50. The variable EH is used to set the Echo to either ON (EH=1) or OFF (EH=0). The program should be modified to read words that are most meaningful to the student by changing or adding DATA statements beginning line 1350.

```
10 REM SCAN WORD FONT
20 REM
30 DIM W$(50), W(50), WX(50)
40 D$ = CHR$(4): G$ = CHR$(5)
50 WP = 5
60 EH = 1
70 HOME
80 VTAB 2: HTAB 12: PRINT "SCAN WORD FONT"
90 VTAB 18
100 PRINT "(PRESS RETURN TO BEGIN OR Esc TO QUIT)";
110 POKE -16368,0: GET R$
120 IF ASC(R$) = 27 THEN VTAB 23: END
130 ONERR GOTO 160
140 N = N+1
150 READ W$(N): GOTO 140
160 POKE 216,0: HOME
170 N = N-1
180 PRINT
190 IF PEEK(999)=99 THEN PRINT D$;"PR#0": GOTO 230
200 POKE 999,99
210 PRINT D$;"BRUN TEXTALKER"
220 FOR D = 1 TO 2000: NEXT D
230 HOME
240 IF PEEK(1000) = 99 THEN 270
250 POKE 1000,99
260 PRINT D$;"BLOAD LASCII"
270 A = PEEK(43634) + PEEK(43635) * 256
280 T = INT(A/256)
290 POKE 232,A-T*256
300 POKE 233,T
310 IF EH = 1 THEN PRINT G$;"O"
320 VTAB 3: HTAB 12: PRINT "SCAN WORD FONT"
330 POKE 216,0: VTAB 22: HTAB 1
340 PRINT "(ENTER CONTROL VALUES AND PRESS RETURN)"
350 VTAB 9: HTAB 1
360 INPUT "SCAN SPEED (1=FAST, 10=SLOW): ";ST$
370 ST = VAL(ST$): IF ST < 1 THEN ST = 1
380 PRINT
390 PRINT "TACHISTOSCOPE (1=ON, 2=OFF): ";TC$
400 VTAB 15
410 PRINT "NUMBER OF WORD READ: "N
420 VTAB 22: HTAB 1: CALL -868
430 VTAB 22: HTAB 6
440 PRINT "(PRESS SWITCH OR KEY TO BEGIN";
450 GOSUB 460: GOTO 490
460 IF PEEK(-16286) > 127 OR PEEK(-16287) > 127 THEN RETURN
470 IF PEEK(-16384) > 127 THEN RETURN
480 GOTO 460
490 POKE -16368,0: HOME
500 RW = RND(-PEEK(78)*1000+PEEK(79))
510 FOR X = 1 TO N
520 RW = INT(RND(1)*N+1)
530 IF W(RW) = 1 THEN 520
```

```
540 W(RW) = 1
550 NA = NA+1
560 HGR2: HCOLOR=7
570 SCALE=1
580 P$(1)=W$(RW)
590 FOR K = 2 TO WP
600 RN = INT(RND(1)*N+1)
610 IF WX(RN) = 1 OR W$(RN) = P$(1) THEN 600
620 WX(RN) = 1
630 P$(K) = W$(RN)
640 NEXT K
650 V = 10: H = 100
660 W$ = W$(RW): GOSUB 700
670 V = 25: H = 5
680 W$ = "-------------------": GOSUB 700
690 GOTO 770
700 FOR L = 1 TO LEN(W$)
710 C$ = MID$(W$,L,1)
720 C = ASC(C$)
730 DRAW C AT H,V
740 H = H+13
750 NEXT L
760 RETURN
770 IF EH = 0 THEN 790
780 PRINT G$"T "W$(RN)" " G$"O"
790 IF TC$ = "2" THEN 830
800 FOR D = 1 TO 500*ST: NEXT D
810 HGR2
820 HCOLOR=7
830 V = 25
840 FOR WA = 1 TO WP
850 RN = INT(RND(1)*WP+1)
860 IF A(RN)=1 THEN 850
870 A(RN) = 1: PC(WA) = RN
880 IF RN = 1 THEN CP = WA
890 V = V+25: H = 100
900 W$ = P$(RN): GOSUB 700
910 NEXT WA
920 POKE -16368,0
930 V = 50: H = 75
940 SP = 1: NS = 1
950 HCOLOR=7
960 DRAW 31 AT H,V
970 IF EH = 0 THEN 990
980 PRINT G$"T "P$(PC(SP))" "G$"O"
990 FOR D = 1 TO ST*10
1000 IF PEEK(-16286) > 127 OR PEEK(-16287) > 127 THEN 1070
1010 KY = PEEK(-16384): IF KY > 127 THEN 1070
1020 NEXT D
1030 HCOLOR=0: DRAW 31 AT H,V
1040 V = V+25: IF V > 25+25*WP THEN V = 50
1050 SP = SP+1: IF SP > WP THEN SP = 1
1060 GOTO 950
1070 POKE -16368,0
1080 IF KY = 155 THEN 1200
1090 V = 25+SP*25: H = 100
1100 W$ = P$(PC(SP))
1110 HCOLOR=0: GOSUB 700
1120 H = 120: HCOLOR=7: GOSUB 700
1130 FOR D = 1 TO 750: NEXT D
1140 IF SP = CP THEN 1160
1150 GOTO 1200
```

```
1160 W$ ="*--->": H = 20: GOSUB 700
1170 IF EH = 0 THEN 1190
1180 PRINT G$"T CORRECT!"G$"O"
1190 SC = SC+1
1200 POKE -16368,0
1210 FOR D = 1 TO 2000: NEXT D
1220 FOR K = 0 TO WP
1230 A(K) = 0: P$(K)="": WX(K)=0
1240 NEXT K
1250 NEXT X
1260 TEXT: HOME
1270 VTAB 7
1280 PRINT "NUMBER = "N
1290 PRINT "ATTEMPTED = "NA
1300 PRINT "CORRECT = "SC
1310 END
1320 REM
1330 REM DATA STATEMENTS
1340 REM
1350 DATA HAMBURGER,TV,RADIO,FUNNY,HELP
```

Sentence Cloze

As a general rule, as the task complexity of a program increases so does the amount of programming involved. The SENTENCE CLOZE program is fairly long because sentences must not only be read but also evaluated in order to select the correct answer as well as item alternatives.

When SENTENCE CLOZE is run, up to 50 sentences (or any string of alphanumeric characters for that matter) are read from DATA statements. The sentences are randomly selected for presentation. For each sentence presented, the word to be deleted from the sentence is specified in the DATA statements as are the incorrect alternatives.

The scanning routine in this program does not use a cursor, but highlights each alternative being scanned by displaying the alternative in INVERSE mode.

```
10 REM SENTENCE CLOZE
20 REM
30 DIM Q$(50),Q(50)
40 ONERR GOTO 80
50 N = 1
60 READ Q$(N)
70 N = N+1: GOTO 60
80 N = N-1: POKE 216,0
90 ST = 4
100 FOR Q = 1 TO N
110 RQ = INT(RND(1)*N+1)
120 IF Q(RQ) = 1 THEN 110
130 Q(RQ) = 1
140 HOME
150 P$(0) = "": QP = 0
160 FOR L = 1 TO 1000: NEXT L
170 PRINT "SENTENCE #"Q
180 VL = 5
190 VTAB VL
200 FOR C = 1 TO LEN(Q$(RQ))
210 L$ = MID$(Q$(RQ),C,1)
220 IF L$ = "/" THEN 260
230 P$(QP) = P$(QP)+L$
```

```
240 IF C = LEN(Q$(RQ)) THEN 260
250 GOTO 280
260 QP = QP+1: P$(QP) = ""
270 IF QP = 7 THEN 290
280 NEXT C
290 QP = QP-1: HTAB 1
300 FOR C = 1 TO LEN(P$(0))
310 L$ = MID$(P$(0),C,1)
320 IF L$ = " " THEN 370
330 IF L$ = "=" THEN L$ = "______"
340 U$ = U$+L$
350 IF C = LEN(P$(0)) THEN 370
360 GOTO 440
370 IF LEN(U$)+P+1 > 39 THEN 410
380 U$ = U$ + " ": PRINT U$;
390 P = P + LEN(U$)
400 U$ = "": GOTO 440
410 PRINT: PRINT U$" ";
420 P = LEN(U$)+1: U$ = ""
430 VL = VL+1
440 NEXT C: P = 0
450 VL = VL+2
460 FOR K = 1 TO QP
470 A(K) = 0: NEXT K
480 FOR QA = 1 TO QP
490 RN = INT(RND(1)*QP+1)
500 IF A(RN) = 1 THEN 490
510 A(RN) = 1: PC(QA) = RN
520 IF RN = 1 THEN CP = QA
530 VTAB VL+QA*2: HTAB 10
540 PRINT P$(RN)
550 NEXT QA
560 SP = 1
570 INVERSE
580 VTAB VL+SP*2: HTAB 10
590 PRINT P$(PC(SP)): NORMAL
600 FOR D = 1 TO ST*20
610 IF PEEK(-16287) > 127 THEN 680
620 IF PEEK(-16384) = 155 THEN 800
630 NEXT D
640 VTAB VL+SP*2: HTAB 10
650 PRINT P$(PC(SP))
660 SP = SP+1: IF SP < = QP THEN 570
670 SP = 1: GOTO 570
680 VTAB VL+SP*2: HTAB 1: CALL -868
690 INVERSE
700 VTAB VL+SP*2: HTAB 12
710 PRINT P$(PC(SP)): NORMAL
720 FOR L = 1 TO 500: NEXT L
730 IF SP = CP THEN 750
740 GOTO 780
750 VTAB VL+CP*2: HTAB 1
760 INVERSE
770 PRINT "CORRECT!": NORMAL
780 FOR L = 1 TO 2000: NEXT L
790 NEXT Q
800 POKE -16368,0
810 VTAB 23: END
820 DATA WHAT=IS IT?/TIME/SAW/GO/LUNCH
830 DATA HE IS A VERY TALL=./MAN/GIRL/WOMAN/SHORT
840 DATA I=HUNGRY./AM/IS/ARE/ME
```

Each DATA statement is comprised of three parts: STEM, CORRECT ALTERNATIVE and INCORRECT ALTERNATIVES. Each DATA statement part is separated by a slash (/). The order for entering items is always to enter the STEM first, followed by the correct answer, and then followed by the incorrect alternatives. When the item is displayed, the program automatically randomizes the alternatives. The equal (=) symbol in the stem component signifies where a space is inserted in place of the missing word when the stem is displayed on screen.

The following illustrates how to add additional sentences to the program:

```
850 THE=BOAT IS IN THE WATER./BOAT/PLANE/CAR/TRAIN
```

Because each item is scanned using inverse mode, the program must be modified in order to use lowercase letters in that there are no lowercase letters when characters are displayed in inverse mode. This is accomplished by using a cursor to scan each choice.

```
580 VTAB VL+SP*2: HTAB 6
590 PRINT "  ": NORMAL
640 VTAB VL+SP*2: HTAB 6
650 PRINT "  "
690
```

With the above modifications in place, both upper- and lowercase letters can be used in sentence items as shown by the following spelling item:

```
860 That=is made of brick./building/bildin/bilden/billding
```

Of course, if a II+ is being used, the use of lowercase characters is not be possible without 80-column card.

In order to provide summary feedback, the number of correct answers can be tallied and summarized after the task has been concluded. The variable CR is used to tally correct answers by adding line 775:

```
775  CR = CR+1
```

And the number of correct items can then be displayed at the conclusion of the end of the program:

```
801 PRINT
802 PRINT "NUMBER OF ITEMS = "N
803 PRINT "NUMBER CORRECT = "CR
```

To determine the percent of correct answers, add variable IA (items attempted) in line 105, and then add lines 804 and 805:

```
105 IA = IA+1
804 PRINT "ITEMS ATTEMPTED = "IA
805 PRINT "PERCENT CORRECT = "INT(CR/IA*100+.5)
```

A Column Matching Task

The following program entitled COLUMN MATCHING demonstrates show the single switch concept can be applied to a variety of assessment tasks. This program reads up to 20 pairs of words and then creates a single switch column matching task. Consider the following pairs of words: CAR/AUTOMOBILE, COLOR/WHITE, LAUGH/SMILE, EARTH/PLANET, READ/BOOK.

After these words had been read from DATA statements (beginning in line

1100), the words are displayed in the format shown below. Because the words are randomly arranged, the exact position of the words changes each time the program is run.

```
█     CAR         A PLANET
      EARTH       B WHITE
      READ        C BOOK
      COLOR       D SMILE
      LAUGH       E AUTOMOBILE
```

Each stimulus item in the left-hand column is first identified by the cursor. While each stimulus item is being identified, the alternative items in the right-hand column are scanned by highlighting each corresponding alternative item letter in sequential order. After a match is found for the first stimulus item in the left-hand column, the letter corresponding to the alternative selected is displayed next to the stimulus item. The symbol > indicates that an alternative item has already been chosen as a match.

```
    E CAR         > PLANET
    A EARTH       > WHITE
    C READ        > BOOK
    B COLOR       D SMILE
█     LAUGH       > AUTOMOBILE
```

After the last item has been presented, answers are checked and summary feedback given. This is not an easy task, but for some students it can provide a useful and interesting variation to the traditional form of single switch scanning task.

The pairs of words are read from DATA statements beginning in line 1100. The program reads up to 20 pairs of words. If fewer than 20 pairs are used (and it might be wise to begin with three or four pairs when first presenting this task), the program automatically adjusts itself to scan the exact number of items specified.

```
10 REM COLUMN MATCHING
20 REM
30 HOME
40 L$ = "MATCHING"
50 VTAB 2: HTAB 20-LEN(L$)/2
60 SM = 5
70 PRINT L$
80 VTAB 22: HTAB 3
90 PRINT "(ENTER SCAN SPEED AND PRESS RETURN)"
100 VTAB 9: HTAB 1
110 INPUT "SCAN SPEED: (1=FAST, 10=SLOW): ";ST$
120 IF ST$ < "1" THEN ST$="1"
130 ST = VAL(ST$)
140 VTAB 15: HTAB 13
150 PRINT "(READING ITEMS)"
160 M = 20
170 DIM S$(M),SS$(M),R$(M),RR$(M),
    V(M),RF$(M),C$(M),A$(M)
180 ONERR GOTO 270
190 N = 1
200 READ S$(N), R$*N)
210 S$(N) = LEFT$(S$(N),16)
220 R$(N) = LEFT$(R$(N),16)
230 IF N = 20 THEN 290
240 N = N+1
250 IF N > 15 THEN N = 15: GOTO 280
260 GOTO 200
```

```
270 N = N+1
280 POKE 216,0
290 R = RND(-PEEK(78)*1000+PEEK(79))
300 FOR K = 1 TO N
310 R = INT(RND(1)*N)+1
320 IF V(R) = 1 THEN GOTO 310
330 V(R) = 1
340 SS$(X) = S$(R)
350 RR$(X) = R$(R)
360 NEXT X
370 FOR K = 1 TO N
380 V(K) = 0
390 NEXT K
400 DOE X = 1 TO N
410 R = INT(RND(1)*N)+1
420 IF V(R) = 1 THEN 410
430 V(R) = 1
440 C$(X) = CHR$(64+R)
450 RF$(R) = RR$(X)
460 NEXT X
470 VTAB 4: CALL -958: VTAB 5
480 FOR K = 1 TO N
490 PRINT TAB(3); SS$(K); TAB(22);
500 PRINT CHR$(64+K); TAB(24); RF$(K)
510 NEXT K
520 X = 1: Y = 1: POKE -16368,0
530 VTAB X+4: HTAB 1
540 INVERSE: PRINT " ": NORMAL
550 VTAB Y+4: HTAB 22
560 INVERSE: PRINT CHR$(64+Y)
570 NORMAL
580 FOR D = 1 TO ST*SM
590 IF PDL(0) < 20 THEN 680
600 IF PEEK(-16286) > 127 OR PEEK(-16287) > 127 THEN
    680
610 KY = PEEK(-16384): IF KY > 127 THEN 680
620 NEXT D
630 VTAB Y+4: HTAB 22: PRINT CHR$(64+Y)
640 Y = Y+1: IF Y > N THEN Y = 1
650 VTAB Y+4: HTAB 22
660 INVERSE: PRINT CHR$(64+Y): NORMAL
670 GOTO 580
680 POKE -16368,0
690 IF KY = 155 THEN 830
700 IF KY = 139 THEN 730
710 A = 64+Y: A$(X) = CHR$(64+Y)
720 GOTO 770
730 VTAB X+4: HTAB 1: PRINT "  "
740 VTAB X+4: HTAB 1: PRINT A$(X)
750 X = X-1: IF X < 1 THEN X = 1
760 GOTO 530
770 VTAB X+4: HTAB 1: PRINT A$(X)
780 VTAB A-60: HTAB 21: PRINT ">"CHR$(64+Y)
790 X = X+1: Y = 1
800 FOR L = 1 TO 500: NEXT L
810 IF X > N THEN 830
820 GOTO 530
830 VTAB 23: HTAB 2
840 PRINT "(CORRECT ANSWERS WILL BE SHOWN ABOVE)"
850 FOR L = 1 TO 1500: NEXT L
860 FOR K = 1 TO N
870 HTAB 22: CALL -868
```

```
880 IF C$(K) < > A$(K) THEN 910
890 C = C+1
900 INVERSE
910 VTAB K+4: HTAB 22
920 PRINT C$(K);" ";RR$(K);
930 CALL -868: PRINT: NORMAL
940 NEXT K
950 VTAB 23: CALL -868: VTAB 23
960 PRINT "NUMBER CORRECT = "C
970 HTAB 25: PRINT "(PRESS RETURN)";
980 POKE -16368,0: GET R$
990 VTAB 4: CALL -958
1000 VTAB 5: HTAB 1
1010 PRINT "PERFORMANCE SUMMARY:"
1020 PRINT
1030 PRINT "NUMBER OF ITEMS = "N
1040 PRINT "NUMBER CORRECT = "C
1050 PRINT "PERCENT CORRECT = "INT(C/N*100+.5)
1060 END
1070 REM
1080 REM DATA STATEMENTS
1090 REM
1100 DATA HORSE,ANIMAL
1110 DATA BIG,LARGE
1120 DATA CAR,AUTOMOBILE
1130 DATA COLOR,WHITE
1140 DATA SPEAK,TALK
1150 DATA LAUGH,SMILE
1160 DATA EARTH,PLANET
1170 DATA RUN,WALK
1180 DATA FORK,SPOON
1190 DATA READ,BOOK
```

Reading Comprehension

Switch Reader

If a student has developed sufficient reading skills, the time is right to begin considering various single switch word processing approaches that can be used to developed reading and writing skills. Needless to say, a single switch word processor requires a rather extensive program, but a single switch system for presenting sentences, passages, and stories can be developed that is not overly complex and very easy-to-use.

The sentences for the SWITCH READER are contained in DATA statements beginning in line 660. Notice that each DATA statement begins with quotation marks. If commas are not used, this is not necessary. However, if a comma is used as the case in lines 690 and 700, the DATA statement must begin with quotation marks so that the entire line is read as one string.

```
10 REM SWITCH READER
20 REM
30 DIM L$(75)
40 N = 1
50 ONEPR GOTO 80
60 READ L$(N)
70 N = N+1: GOTO 60
80 N = N-1: POKE 216,0
90 HOME
100 VTAB 2: HTAB 11
```

```
110 PRINT "SINGLE SWITCH READER"
120 VTAB 7
130 INPUT "SCAN SPEED (1=FAST, 9=SLOW): ";ST$
140 ST = VAL(ST$): IF ST < 1 THEN ST = 5
150 FOR K = 1 TO 40
160 E$ = E$+" ": NEXT K
170 C$(1) = "<": C$(2) = ">"
180 P = 1: SP = 1
190 HOME
200 INVERSE
210 VTAB 3: PRINT E$
220 VTAB 19: PRINT E$
230 NORMAL
240 VTAB 2: PRINT "LINE = "LN
250 VTAB 2: HTAB 15: PRINT "PAGE = "P" "
260 VTAB 2: HTAB 31: PRINT "TEXT = "N
270 FOR K = 1 TO 2
280 VTAB 22: HTAB 17+(K+1)*6
290 IF SP = K THEN INVERSE
300 PRINT C$(K): NORMAL: NEXT K
310 IF PEEK(-16287) > 127 THEN 310
320 FOR D = 1 TO ST*10
330 IF PEEK(-16287) > 127 THEN 380
340 KY = PEEK(-16384): IF KY = 155 THEN 600
350 NEXT D
360 SP = SP+1: IF SP > 2 THEN SP = 1
370 GOTO 240
380 IF SP = 2 THEN 500
390 LN = LN-1: IF LN < 1 THEN 410
400 GOTO 240
410 LN = 7: P = P-1: IF P < 1 THEN P = 1
420 FOR K = 5 TO 17
430 VTAB K: PRINT E$: NEXT K
440 LN = 7
450 FOR K = 7 TO 1 STEP -1
460 SL = (P-1)*7+K
470 VTAB K*2+3
480 PRINT L$(SL): NEXT K
490 GOTO 240
500 LN = LN+1: IF LN > 7 THEN 530
510 GOTO 560
520 LN = 1: P = P+1
530 IF P > 10 THEN P = 1
540 FOR K = 5 TO 17
550 VTAB K: PRINT E$: NEXT K
560 SL = (P-1)*7+LN
570 VTAB LN*2+3
580 PRINT L$(SL)
590 GOTO 240
600 POKE -16368,0
610 END
620 REM
630 REM DATA STATEMENTS
640 REM
650 DATA "THIS IS A STORY ABOUT A DOG. THE DOG'S
660 DATA "NAME IS FRED.  FRED IS A FUNNY DOG.
670 DATA "FRED TALKS!
680 DATA "WHEN FRED IS HAPPY, HE WAGS HIS TAIL
690 DATA "WHEN FRED IS HUNGRY, HE BANGS HIS BOWL
```

To enter data directly from a text file, make these modifications:

```
45 D$ = CHR$(4)
46 F$ = "STORY"
47 HOME
55 PRINT D$;"OPEN"F$
56 PRINT D$;"READ"F$
60 INPUT L$(N)
80 POKE 216,0
85 PRINT D$;"CLOSE"F$
```

To view the file as it is being read from disk, enter the following:

```
52 PRINT D$;"MON C,I,O"
81 GET KY$
```

Press RETURN to begin the program after the file items have been read.

Text File Maker

The above modification is not very useful if there is no way to create a text file. The following short program can be used to create a text file by means of INPUT statements. However, this is an intentionally uncomplicated program so don't expect much in the way of word processing frills.

To use the program, enter the sentence number and then the sentence. To file the sentences (or strings), enter a /; to exit, enter * (see lines 120 and 130). When used with the SINGLE SWITCH READER, sentences should be 40 characters of less.

The CHR$(34) in line 210 is the ASCII equivalent of a quotation marks. This allows the use commas in sentences. However, quotation marks within sentences cannot be used. If quotes are needed, use two single quotation mark (e.g., '' rather than ").

```
10 REM TEXT FILE MAKER
20 REM
30 DIM L$(100)
40 HOME
50 D$ = CHR$(4)
60 F$ = "STORY"
70 PRINT
80 INPUT "SENTENCE #, /=FILE, *=EXIT: ";N$
90 N = VAL(N$)
100 IF N > M THEN M = N
110 IF N > 1 THEN 140
120 IF N$ = "/" THEN 160
130 IF N$ = "*" THEN 250
140 INPUT "";L$(N)
150 GOTO 70
160 PRINT D$;"OPEN "F$
170 PRINT D$;"DELETE "F$
180 PRINT D$;"OPEN "F$
190 PRINT D$;"WRITE "F$
200 FOR K = 1 TO M
210 PRINT CHR$(34)+L$(K)
220 NEXT K
230 PRINT D$;"CLOSE "F$
240 GOTO 80
250 VTAB 23
260 END
```

Text File Reader

Text files are designated by a **T** in the catalog. In order to see the contents of a text file without actually running the program that uses the file, run TEXT FILE READER and enter the text file to be read following the prompt. This program reads each line of the file, stores the line in the string variable L$, and then prints L$. When the program attempts to read beyond the last line of a file, an error is detected and the file is closed in line 170.

```
10 REM TEXT FILE READER
20 REM
30 D$ = CHR$(4)
40 HOME
50 VTAB 5
60 PRINT D$;"MON C"
70 INPUT "FILE NAME: ";F$
80 ONERR GOTO 160
90 PRINT D$;"OPEN ";F$
100 PRINT D$;"READ ";F$
110 PRINT
120 INPUT L$
130 PRINT L$
140 FOR L = 1 TO 50: NEXT L
150 GOTO 120
160 PRINT
170 PRINT D$;"CLOSE ";F$
```

Using Word Processor Files

Although most word processor text files can be accessed, there could be a problem when reading the file because the format for storing text on disk varies from one word processor to the next. With this word of caution in mind, SWITCH READER-2 loads a word processing file to be used as the text source within the program:

```
10 REM SWITCH READER-2
20 REM
30 DIM L$(500)
40 N = 1
50 HOME
60 F$ = "ONE.MW"
70 PRINT CHR$(4);"BLOAD"F$
80 BA = PEEK(43634) + PEEK(43635) * 256
90 HIMEM: BA-1
100 BL = PEEK(43616) + PEEK(43617) * 256
110 FOR K = BA+256 TO BA+BL
120 PK = PEEK(K)
130 IF PK = 13 OR PK = 141 THEN PK = 32
140 C = C+1: IF C > 38 THEN 75
150 GOTO 180
160 C = 0: N = N+1
170 IF PK = 0 THEN 210
180 L$(N) = L$(N)+CHR$(PK)
190 PRINT CHR$(PK);
200 NEXT K
210 HOME
220 VTAB 2: HTAB 13
230 PRINT "SWITCH READER-2"
240 VTAB 7
250 INPUT "SCAN SPEED (1=FAST, 9=SLOW): ";ST$
```

```
260 ST = VAL(ST$): IF ST < 1 THEN ST = 5
270 FOR K = 1 TO 40
280 E$ = E$+" ": NEXT K
290 C$(1) = "<": C$(2) = ">"
300 P = 1: SP = 1
310 HOME
320 INVERSE
330 VTAB 3: PRINT E$
340 VTAB 19: PRINT E$
350 NORMAL
360 VTAB 2: PRINT "LINE = "LN
370 VTAB 2: HTAB 15: PRINT "PAGE = "P" "
380 VTAB 2: HTAB 31: PRINT "TEXT = "N
390 FOR K = 1 TO 2
400 VTAB 22: HTAB 17+(K-1)*6
410 IF SP = K THEN INVERSE
420 PRINT C$(K): NORMAL: NEXT K
430 IF PEEK(-16287) > 127 THEN 430
440 FOR D = 1 TO ST*10
450 IF PEEK(-16287) > 127 THEN 500
460 KY = PEEK(-16384): IF KY = 155 THEN 720
470 NEXT D
480 SP = SP+1: IF SP > 2 THEN SP = 1
490 GOTO 360
500 IF SP = 2 THEN 620
510 LN = LN-1: IF LN < 1 THEN 530
520 GOTO 360
530 LN = 7: P = P-1: IF P < 1 THEN P = 1
540 FOR K = 5 TO 17
550 VTAB K: PRINT E$: NEXT K
560 LN = 7
570 FOR K = 7 TO 1 STEP -1
580 SL = (P-1)*7+K
590 VTAB K*2+3
600 PRINT L$(SL): NEXT K
610 GOTO 360
620 LN = LN+1: IF LN > 7 THEN 640
630 GOTO 680
640 LN = 1: P = P+1
650 IF P > 10 THEN P = 1
660 FOR K = 5 TO 17
670 VTAB K: PRINT E$: NEXT K
680 SL = (P-1)*7+LN
690 VTAB LN*2+3
700 PRINT L$(SL)
710 GOTO 360
720 POKE -16368,0
730 END
```

In the above program, string variable F$ in line 60 is used to specify the name of the word processor file. Remember to use the name of the file that is actually on the disk. Because it is sometimes difficult to determine exactly where the text in a word processor file begins, a bit of experimentation with line 110 might be required.

The **HIMEM:** instruction in line 90 is used to reset memory so that the string variables are stored immediately below the text file rather than at the usual HIMEM starting point of 38400. The starting location of the text file is indicated by variable BA (line 80).

Database Applications

A database can be used to retrieve data for use within a program, or to store data on disk for use or consideration at a later time. The one important feature of most database systems is the ability to change and modify the data. The following section uses the READ program discussed in the last section to illustrate three important database applications:

1) Database Control Variables
2) Text Database Systems
3) Results Database Systems

Although these database systems are used within the context of a particular program, the underlying database principles can be used with many of the program discussed in this manual.

Database Programming

The READ-2 program listed below contains three types of database systems involving control variables, text data, and results. The software required to read/modify these databases are discussed in the following sections.

The actual purpose of READ-2 is similar to that of READ, except that the words are flashed on the screen tachistoscopically as a function of the scan speed which is read from a disk file called CONTROL.SW.

When READ-2 is run, the program reads a file from disk containing up to N words. The words are stored in W$(K) as shown in line 160. Next, the scan speed is read from disk. This option is useful if the scan speed remains relatively constant from one program run to the next, and if a disk contains a variety of programs. Thus, changing the scan speed on disk is all that is necessary to change the scan speed for all the programs using the scan speed database option. If the scan speed has not been set using the scan speed create program called DOS CONTROL, the scan speed is automatically set to 5 (see line 270).

Finally, the results of the program are stored on disk using the DOS APPEND command (line 840). This command adds the results to the text file immediately after the last record (i.e., line of data) in the file.

```
10 REM READ-2
20 REM
30 DIM W$(50),W(50),A(50)
40 HOME
50 VTAB 5
60 INPUT "ENTER FILE NAME: ";F$
70 CR$ = "  "
80 AL = 4
90 D$ = CHR$(4)
100 ONERR GOTO 160
110 PRINT D$;"OPEN "F$
120 PRINT D$;"READ "F$
130 INPUT N
140 FOR K = 1 TO N
150 INPUT W$(K): NEXT K
160 PRINT D$;"CLOSE "F$
170 POKE 216,0
180 VTAB 10: PRINT "ITEMS IN FILE: "N
190 CF$ = "CONTROL.SW"
200 ONERR GOTO 240
210 PRINT D$;"OPEN "CF$
```

```
220 PRINT D$;"READ "CF$
230 INPUT ST$
240 PRINT D$;"CLOSE "CF$
250 POKE 216,0
260 ST = VAL(ST$)
270 IF ST < 1 THEN ST = 5
280 FOR W = 1 TO N: A = A+1
290 RW = INT(RND(1)*N+1)
300 IF W(RW) = 1 THEN 290
310 W(RW) = 1
320 HOME: VL = 7
330 CP = INT(RND(1)*AL+1)
340 A$(CP) = W$(RW)
350 VTAB 5: HTAB 16
360 PRINT A$(CP)
370 FOR L = 1 TO 200*ST: NEXT L
380 VTAB 5: CALL -868
390 FOR K = 1 TO AL
400 IF K = CP THEN 450
410 RN = INT(RND(1)*N+1)
420 IF A(RN) = 1 OR RN = RW THEN 410
430 A(RN) = 1
440 A$(K) = W$(RN)
450 VTAB VL+K*2: HTAB 16
460 PRINT A$(K)
470 NEXT K
480 P = 1
490 INVERSE
500 VTAB VL+P*2: HTAB 13
510 PRINT CR$: NORMAL
520 FOR D = 1 TO ST*30
530 IF PEEK(-16287) > 127 THEN 600
540 KY = PEEK(-16384): IF KY > 127 THEN 600
550 NEXT D
560 VTAB VL+P*2: HTAB 13
570 PRINT CR$
580 P = P+1: IF P < = AL THEN 490
590 P = 1: GOTO 490
600 POKE -16368,0
610 IF P = CP THEN 630
620 GOTO 670
630 VTAB VL+CP*2: HTAB 6
640 INVERSE
650 PRINT "CORRECT!": NORMAL
660 C = C+1
670 FOR L = 1 TO 2000: NEXT L
680 IF KY = 155 THEN 710
690 FOR K = 1 TO N: A(K) = 0: NEXT K
700 NEXT W
710 DF$="RESULTS.DATA"
720 HOME: VTAB 3
730 PRINT "RESULTS:"
740 PRINT
750 PRINT "ITEMS = "N
760 PRINT "ATTEMPTED = "A
770 PRINT "CORRECT = "C
780 PRINT "PERCENT = "INT(C/A*100+.5)
790 DS$ = STR$(N)+" " + STR$(A) + " " + STR$(C)
800 ONERR GOTO 820
810 GOTO 840
820 POKE 216,0
830 PRINT D$;"OPEN "DF$
```

```
840 PRINT D$;"APPEND "DF$
850 PRINT D$;"WRITE "DF$
860 PRINT DS$
870 PRINT D$;"CLOSE "DF$
880 END
```

The first database system is located in lines 100 to 180. The number of words in the file is read from the file via the INPUT statement in line 130 and the words are read into W$(K) in line 150. Line 110 "opens" the file specified in F$. while line 120 gives the instruction to "read" data from the file. After all the words have been read, the file is closed in line 160.

The second database component is located in lines 200 to 270. This file is specified by variable CF$ and contains the scan speed used to determine the amount of time each word is scanned and also the length of time each word is tachistoscopically displayed on screen. The ONERR loop in 200 insures that if there is no scan speed data file, the program loops to line 240 rather than interrupting and ending the program run. The POKE in line 250 deactivates the ONERR function so that if an error occurs elsewhere in the program, control is not automatically shifted to line 240.

Before running the READ-2 program, the READ AUTHOR program is used to create the word or item database. When the program is run, enter the name of the word file containing the words to be used by the program.

The last database system is located in lines 800 to 870. There are several Applesoft DOS methods that can be used to store and retrieve results from disk. The routine used in READ-2 is not elaborate but it does work and can be used to store a large number of data files. The APPEND command in line 840 signifies that the data record is "appended" to the data file; that is, the record is to be stored after the last existing record in DF$. The actual data record is stored as a string by condensing all the relevant summary data to a single variable string. The spaces are included in the string so as to facilitate file interpretation when the data file is retrieved from disk by means of the RESULTS READER program.

If no data exists in the RESULTS.DATA file, a file by this name is opened (i.e., created) by means of the ONERR routine in lines 800 to 830.

The name of the file in the READ-2 program is preset by

```
60 F$ = "WORDS"
```

The program is set to flash words tachistoscopically by means of lines 370 and 380. This option is deactivated and the word or string presented remains on the screen by deleting these lines 370 and 380.

Creating/Modifying Database Files

The database systems described in the READ-2 program work in conjunction with the following three programs: DOS CONTROL, READ AUTHOR and RESULTS READER. The data contained in these files is stored on disk as text files. If all three systems are used, the disk catalog entries will look something like this:

```
T 002 CONTROL.SW
T 002 WORDS
T 002 RESULTS
```

The second text file can contain up to 75 words or strings. The actual size of the file will vary depending on the number and length of individual words and/or strings. Each word file must be stored on disk using a different file name.

DOS CONTROL: This program creates a text file in which scan speed is stored and then retrieved for use in the READ-2 program. The DOS instruction in line 50 is a monitor command and is used to MONitor Commands to the disk (**C**), Input from the disk (**I**) and Output to the disk (**O**). To turn off the MON command, the following is used:

```
PRINT D$;"NOMON C,I,O"
```

The current scan speed in CONTROL.SW is retrieved by lines 80 to 140. After the new scan speed has been specified (line 200), the scan speed is stored in CONTROL.SW.

Before entering the new scan speed in the file, the file is deleted and then re-opened for the new scan speed in line 240. To enter data into a text file, the DOS Write command is given in line 250. Data is actually written into the file by means of PRINT statements (line 260).

```
10 REM DOS CONTROL
20 REM
30 HOME
40 D$ = CHR$(4)
50 PRINT D$;"MON C,I,O"
60 PRINT
70 ONERR GOTO 140
80 F$ = "CONTROL.SW"
90 PRINT D$;"OPEN "F$
100 PRINT D$;"READ "F$
110 INPUT ST$
120 ST = VAL(ST$)
130 PRINT D$;"CLOSE "F$
140 POKE 216,0
150 HOME: HTAB 3
160 PRINT: PRINT: PRINT "SCAN SETTING:"
170 PRINT
180 PRINT "CORRECT SETTING: "ST$
190 PRINT: PRINT
200 INPUT "SCAN SPEED (1=FAST TO 10=SLOW): ";ST$
210 PRINT: PRINT
220 PRINT D$;"OPEN "F$
230 PRINT D$;"DELETE "F$
240 PRINT D$;"OPEN "F$
250 PRINT D$;"WRITE "F$
260 PRINT ST$
270 PRINT D$;"CLOSE "F$
280 END
```

READ AUTHOR: The READ AUTHOR program is an authoring system for creating text files in which the file can have as many as 75 items, and each item can be up to 28 characters in length. An item can be a word, characters, phrase or even mathematical statements (e.g., 3 + 12 + 9 = 24).

As indicated by several of the lines in the following program, the actual code for an authoring system can be fairly complex. Without going into too much detail, the heart of the program is line 340 which reads keyboard input character-by-character using variable C$. The CHR$ statements determine whether an arrow key or the RETURN key has been pressed, or whether a line has been deleted (line 390) or the filing routine initiated (line 360). Note that in line 360, CHR$(6) is equivalent to CONTROL+F which is the command used to file the text.

```
10 REM READ AUTHOR
20 REM
30 VS = 75
40 DIM W$(VS)
50 HOME: VTAB 2: HTAB 10
60 FOR K = 1 TO 40: H$ = H$+"-": NEXT K
70 PRINT "READING AUTHORING SYSTEM"
80 D$ = CHR$(4)
90 VTAB 4: PRINT H$
100 VTAB 20: PRINT H$
110 VTAB 7
120 INPUT "FILE NAME: ";F$
130 ONERR GOTO 200
140 PRINT D$;"OPEN "F$
150 PRINT D$;"READ "F$
160 INPUT N
170 FOR K = 1 TO N
180 INPUT W$(K): NEXT K
190 PRINT D$;"CLOSE "F$
200 POKE 216,0
210 VTAB 7: CALL -868
220 VTAB 21: HTAB 1
230 PRINT "CONTROL+F = QUIT, CONTROL+X = ERASE LINE";
240 PRINT "  ---USE ARROW KEYS TO MOVE CURSOR---";
250 PRINT: P = 1: VN = 1
260 PRINT
270 FOR K = P TO P+14
280 LN = LN+1: VTAB LN+4: HTAB 1
290 PRINT "WORD ";
300 IF K < 10 THEN PRINT SPC(1);
310 PRINT K": "W$(K);: CALL -868
320 NEXT K: LN = 0
330 VTAB VN+4: HTAB 10+HT
340 POKE -16368,0: GET C$
350 IF C$ > CHR$(31) THEN 450
360 IF C$ = CHR$(6) THEN 670
370 IF C$ = CHR$(8) THEN 530
380 IF C$ = CHR$(11) THEN 630
390 IF C$ < > CHR$(24) THEN 420
400 W$(P+VN-1) = "": HT = 0
410 VTAB VN+4: HTAB 10: CALL -868: GOTO 330
420 IF C$ = CHR$(13) OR C$ = CHR$(10) THEN 590
430 IF C$ = CHR$(21) THEN HT = HT+1: IF HT > 30 THEN HT =
    30
440 GOTO 330
450 IF HT = 28 THEN 340
460 IF HT > 0 THEN 490
470 VTAB VN+4: HTAB 10: CALL -868
480 VTAB VN+4: HTAB 10: W$(P+VN-1) = ""
490 W$(P+VN-1) = W$(P+VN-1) + C$
500 PRINT C$;: HT = HT+1
510 POKE 216,0
520 GOTO 340
530 IF LEN(W$(P+VN-1)) < 2 THEN W$(P+VN-1) = "": GOTO 550
540 W$(P+VN-1) = MID$(W$(P+VN-1), 1, LEN(W$(P+VN-1)) - 1)
550 VTAB VN+4: HTAB 10
560 PRINT W$(P+VN-1);: CALL -868
570 HT = HT-1: IF HT < 0 THEN HT = 0
580 GOTO 330
590 VN = VN+1: IF VN = 16 THEN VN = 1: GOTO 610
```

```
600 HT = 0: GOTO 330
610 P = P+15: HT = 0: IF P > VS-14 THEN P = VS-14
620 GOTO 260
630 VN = VN-1: IF VN = 0 THEN VN = 15: GOTO 650
640 HT = 0: GOTO 330
650 P = P-15: HT = 0: IF P < 1 THEN P = 1
660 HT = 0: GOTO 260
670 N = 1
680 FOR K = 1 TO VS
690 IF W$(K) = "" THEN 710
700 W$(N) = W$(K): N = N+1
710 NEXT K: N = N-1
720 NN = N
730 HOME: VTAB 5
740 PRINT D$;"OPEN "F$: PRINT D$;"DELETE "F$
750 PRINT D$;"OPEN "F$: PRINT D$;"WRITE"F$
760 PRINT N
770 FOR K = 1 TO N
780 PRINT W$(K): NEXT K
790 PRINT D$;"CLOSE "F$
800 END
```

RESULTS READER: Programs which are used to access and read database results can be extremely involved and can contain a great many options. The RESULTS READER program illustrates a simple technique for retrieving data from disk. The program reads the various records in the database file, and lists each record immediately after it has been read. The data within each record can be further analyzed by separating the three components comprising each record (number of items, number attempted, and number correct) by means of the space separating each record component.

```
10 REM RESULTS READER
20 REM
30 HOME: VTAB 3
40 D$ = CHR$(4)
50 F$ = "RESULTS"
60 PRINT F$" DATABASE: "
70 PRINT
80 ONERR GOTO 160
90 PRINT D$;"OPEN "F$
100 PRINT D$;"READ "F$
110 INPUT SD$
120 N = N+1
130 PRINT "#"N" "SD$
140 PRINT
150 GOTO 110
160 PRINT D$;"CLOSE "F$
170 POKE 216,0
180 END
```

References

Apple II Reference Manual. Cupertino, California: Apple Computer Inc., 1981.

Apple II DOS Manual. Cupertino, California: Apple Computer Inc., 1981.

Biklen, D. Communication unbound: autism and praxis, **Harvard Educational Review**, 1990, 60, 291-314.

Burns, E. **TRS-80 Teaching Aid**. Reston, Virginia: Reston, 1981.

Burns, E. **The Apple Math and Reading Development Kit**. Reston, Virginia: Reston, 1985.

Burns, E. AUTOSCAN: An adaptive multiple-choice BASIC computer program. **Journal of School Psychology**, 1988, 26, 311-315.

Burns, E. A primary fact scan program, **Closing the Gap**, 1989, 7, 22-23; Single switch letter identification, 1990, 8, 10-11; Vertical scan speech (VSS), 1990, 9, 16-17.

Burns, E. and Mistrett, S. Switchware technology: improving computer usage for students with physical or motor impairments. **Educational Technology**, 1989, 28, 45-47.

Burns, E. and Mistrett, S. Assessing and modifying random keyboard behavior. **Closing the Gap**, 1988, 6, 22-23.

Clute, A. and Eddy, J. **Touch Window Toolkit: Apple II Version - Rev. 1.0**. San Jose, California: Personal Touch Corporation, 1985.

Apple II ProDOS User's Manual. Cupertino, California: Apple Computer Inc., 1983.

Applesoft Tutorial. Cupertino, California: Apple Computer Inc., 1981.

Applesoft BASIC Programming Reference Manual. Cupertino, California: Apple Computer Inc., 1981.

Echo IIb Speech Synthesizer: Santa Barbara, California: Street Electronics Corporation, 1986.

Haskell, Richard. **Apple BASIC**. Englewood Cliffs, New Jersey: Prentice-Hall, 1982.

IBM BASIC Reference Manual. International Business Machines, 1984.

Inman, Don and Inman, Kurt. **Apple Machine Language**. Reston, Virginia: Reston, 1981.